# Open Source Software Development Patterns and License Terms

Alexandre Gaudeul
University of Toulouse

September 17, 2004

**Abstract**

This paper examines the choice of license terms along the development of a piece of software. Three licenses are compared, the proprietary one, the Berkeley Software Distribution, and the General Public License. The choice of one or the other license depends on the characteristics of the software's user base, the market conditions on the developers' job market and the costs involved in maintaining a proprietary software vs. the costs involved in coordinating a software project in a decentralized fashion. That choice influences the distribution of welfare between users, developers and the software's development leader. It also determines the software's pace of development and thus the level of welfare generated. The model explains why a software's license terms may change along its development. Several scenarii may arise, depending on the initial conditions and the chance events along the life of the project. In the context of this paper, open-source license terms are chosen even when they result in a reduction in global welfare. Welfare is increased by forbidding the use of the GPL license terms and going back to the alternative between proprietary and public domain licenses.

# 1 Introduction

This paper models the development of a piece of software to analyze the differences between the main software license terms – proprietary, Berkeley Software Distribution ("BSD") and General Public License ("GPL"). The choice by developers of one or the other software license will be analyzed. The paper relates the choice of software license terms to economic variables that can be estimated empirically. Because license terms are closely related to software development practices, the paper thus allows to make predictions about their evolution. The paper ends with some reflections on the welfare consequences of the existence of open-source software ("OSS") licenses.

**Modeling** Software development is modeled as a two-stage process, the generation of an idea for development and its implementation. An initial software developer, the 'project leader' or 'the innovator' begins a software project alone. She then needs others to help her generate ideas and implement them so as to finish her project. She must decide under what license term her software will be released for others to work on. There are three types of licenses that a developer can adopt when distributing a piece of software. Proprietary license terms forbid any independent change or redistribution of the source code of the program. The GPL allows such change or redistribution, but the modified or redistributed software must be released under the same GPL. The BSD gives even more freedom than the GPL, as modified versions or copies of the program can be redistributed under any license. The license choice determines different development patterns for the software. Proprietary software will remain under the control of its author and proprietary license terms allow the initial developer to exploit her software commercially by selling it on the market. GPL software will remain GPL and any modification will be made public for free. BSD software may become GPL or Proprietary over its life, and thus adopt different development patterns. The difference between the proprietary, BSD and GPL license terms is modeled as a set of restrictions on the actions developers who work on a project are allowed to take. The initial developer has the choice between the proprietary license terms, the BSD and the GPL. A developer who then works on the project will not be able to change the software's license terms if the proprietary or the GPL license was chosen. If the BSD was chosen, she can change the license to the proprietary one, to the GPL or keep it BSD. This is because the BSD license terms allows anybody to change the software's license terms.

After presenting the model's main characteristics, a brief analysis of its dynamics will be made before moving on to some insights it provides. Empirical evidences supporting the model's assumptions will be provided, after which the literature on the topic will be presented. A discussion of the model's limitations is postponed to part 6. The paper concludes on the desirability of encouraging the use of OSS licenses from a public policy point of view.

The two steps in software development, idea generation and implementation, are modeled as probabilities $\lambda$ and $\mu$ that a developer will get an idea for the development of the project and implement it, respectively. There cannot be an implementation if there is no existing idea, so that an idea

must be generated first. Generating an idea is different from implementing it because an idea cannot lead to commercial exploitation while an implementation can. Licenses can only protect the expression of an idea, not the idea itself. This means there is no legal protection for ideas, and somebody who considers making a commercial exploitation of a software must keep her development idea secret or share it under a confidentiality agreement (employment contract). When she then puts the software on the market, being first means she will be able to get most of the commercial value of the software even if there is no way to prevent others to imitate the idea; the cost and delay involved in implementing the same idea in an original way so as not to infringe the copyright of the first to market are too high to make it profitable. The model assumes that the initial project developer stops any involvement in the project's idea generation/implementation game from its outset on, either because she reached the limits of her capacities or wants to work on another project. There is a pool of developers who face cost $c$ to work on the program, and can be hired at cost $w$ to work on it. The interpretation of $w$ as a wage will be employed throughout the model for simplicity, but as said later, $w$ can have other meanings. There is no way to select programmers based on their abilities, and a hired developer doesn't perform better or worse than a developer who works on the software for free or out of personal interest. The initial developer is assumed to derive a higher value $U$ from the eventual completion of the software project than the value $v$ derived by other software developers and users. The completed software has value $V$ on the market, and that value can be appropriated only if the software is proprietary – meaning it cannot be copied freely. Users and developers derive the same benefits $v$ from the software, whether they worked on its development or not.

**Preliminary analysis** There is a conflict of interest between the ideal software license for the initial developer and for subsequent ones. All want the software to be developed as fast as possible so as to be able to use it; but the initial developer would like to be able to sell it to the users/developers, which is possible only if it is developed under proprietary license terms. Users/developers would prefer to see it developed under open-source terms, which allows them to get the software for free. They also prefer the BSD to the GPL because the BSD allows them to make a commercial use of any work they performed on the software. The innovator prefers the GPL to the BSD though, because under the GPL, she will get any implementation for free. Under the GPL however, developers may lack the motivation to work on the software because there is no way to commercially exploit it. The initial developer may therefore want to make her software public under the BSD license term. Indeed, this allows subsequent developers to exploit commercially their own work, thus enhancing their incentives.

The initial developer will thus choose the GPL instead of the proprietary license terms if $w$ is too high (or alternatively, $V$ too low) for a commercial software to be of much interest. $c$ must be low enough compared to $v$ to provide sufficient incentives for developers to work on the project. It is sufficient that $w \geq c$ for a developer to work on the software if it is under proprietary license terms. Working on a GPL project, on the other hand, must lead with sufficiently high probability to the project being implemented and providing value $v$ to the developer so as to cover her cost $c$

of working on it. The innovator will choose the BSD when $c$ and $w$ are too high for the GPL or the proprietary license terms to be interesting.

The BSD license terms may encourage faster development than all other types of licenses; unlike the proprietary license terms, many developers may come to work on the project in a race to implementation and commercialization. Under proprietary license terms, only one developer, the initial one, is interested in getting the software developed. Software under the GPL license terms does not see that race to implementation either because the GPL encourages the sharing of ideas. Under the BSD, a developer may choose to work on the project, keep her idea for development secret, and hire a developer at cost $w$ to work on it under proprietary license terms. This doesn't happen under the GPL as that license cannot be changed and any implementation must thus be made available to others at no charge. This means there is no motivation for parallel development. The BSD license terms encourage developers to conceal their idea and thus leads to many different ideas being under proprietary development at the same time. While this may lead to unnecessary duplication of effort, the model shows that under some conditions, where developers choose to work on a BSD project only once they know no other commercial implementation has been released before, the welfare generated under the BSD is always higher than under the GPL. There is another effect at work that goes against the GPL: a developer can choose to free-ride on the efforts of others, waiting for another developer to work on the project and make it available for free, instead of working on it now. That effect is not present under the BSD because not working on the project now means somebody else will make a commercial implementation of it that you will have to pay to use.

Proprietary license terms are shown to almost always be chosen when $w = c$. This means the use of open-source license may be explained by the high costs involved in maintaining a proprietary software firm and hiring developers, compared to the cost $c$ to develop a piece of software. Those may be legal and organizational costs, but another interpretation is that they are wages. Those wages may be inflated compared to the software developers' costs of working on the project due to imbalances in the software developers' job market. Evidences will be given for both interpretations, but the one adopted here is that $w$ is a wage.

**Main assumptions** The model makes two types of distinctions between developers. The first one is between the leader(s) and the followers, the second one is between innovators and implementers. Those distinctions are not of the same order. The model assumes there exists a leader for the software project – the one who owns the original code, and decides on its license terms. Then, there are followers – those who come to work on the leader's project. Every follower can with some probability be an innovator – have an idea of where to lead the project – or an implementer – know how to implement an existing idea.

The present model assumes that the software is at a relatively advanced stage after having been developed initially by one or a group of developer – the leaders. It does not attempt to explain the

origin of the software; it is simply assumed to exist.[1] Gaudeul (2003), a case study on TEX, an open source typesetting software, and various other case studies on open source software projects, show the same pattern in the emergence of an open source software project: A software program is developed by a closed group of individuals, most of the time one developer only, who then decides, once the project is relatively advanced, to show it to other people in the programmers' community. At that time, the software may already be functioning satisfactorily for the initial developers, but requires many improvements in order to become of use by the general public. The stage at which software can become used by all is crucial, as network effects are very important for software. The number of potential users for the software determines much of its value, directly as in networking software or indirectly for example because a successful software will keep on being supported and developed. This stage in the development is particularly difficult: most OSS are not used much beyond their developers' community, as testified by the Sourceforge graveyard, where most OSS that are listed do not receive any contribution. The initial developer usually decides to release the code of her software for development by others because she has reached the limits of what she wants the software to do. It is usually not because of a limit in her programming abilities; most open source software projects innovators are exceptional programmers. Nonetheless, most innovators, once they have made their program open source, want to disengage from its development as soon as possible. They organize the nascent project's developers' community in a way that makes the project as independent from the innovator as possible. This is either because most of the potential value of the software to the innovator has already been realized, and she does not have much to gain from further development, or because she has found another more interesting and challenging project to work on. Most of the time, the software needs to be adapted to new computer systems – pcTEX was a commercial implementation of TEX for the PC – or an users' interface must be developed – this is how Scientific Word, which is an user interface to many programs, including TEX, came to be. But the development work can also go into totally new directions: TEX, which was meant to ease mathematical typesetting, had to be changed to output pdf files (pdfTEXproject) or to produce interactive educational material (ConTEXt). Those developments made the software available to new sets of users, or increased its value to the existing set of users.

The second distinction in the model, the one between innovators and implementers, comes from the casual observation that most on-line innovators tend to fall into two categories, 'visionaries' and 'technologists', the first ones being more interested in ideas and others schooled in code. There are developers ("geeks") who like just being given a new idea to work on and getting into its technical aspects of it vs. those ("entrepreneurs/idealists") that speak more about larger concepts and ideas. They also want to involve communities and people in their projects, and not seldom, make money, although other objectives may be present. The distinction that is made between 'geeks' and 'idealists' can be traced back to how Schumpeter (1928, 1934) viewed the process by which new combinations of productive means come about: A new product comes into existence with the

[1] Linux's creator, Linus Torvalds, thus built on the Minix operating system. VIM's creator, Bram Moolenar, began with a older clone of the vi text editor called stevie. Raymond built his Fetchmail e-mail utility program on the open-source popclient.

meeting of an invention and an entrepreneur. The entrepreneur gets the difficult task of evaluating and implementing innovative ideas, those possibly coming from another developer than her. The inventor/innovator/generator has a difficult choice when releasing her invention for others to improve upon: those others may appropriate the invention for their own benefit and may also release it later as their own. This is allowed if they make appropriate, significant changes to ensure that the innovator cannot claim the new release is under her copyright. Then, the innovator not only cannot exploit her invention commercially, but also may have to pay to use its commercial implementation. While this may discourage an innovator from releasing her ideas open-source, Johnson (2003) points out that the same effect may be present in proprietary software companies. Hired programmers' motivation to reveal their ideas for development will also be weak, not only because those will be appropriated by the firm's owner, but also because revealing an idea means more work later to implement it. Career concerns are assumed in this paper to give sufficient motivation to the developers who work on a proprietary product in a firm.

The model assumes that developers face a cost $c$ to work on a project, but a developer who hires another to work for her faces a cost $w$ to do so. The difference in the cost of working on an open-source software project and that of working on a proprietary one is of two orders. Organizational and legal costs differ due to the different structure of both development organizations. Proprietary firms face a cost to motivate hired developers to cooperate and accept directions in a firm, while in OSS projects, everybody is free to choose its own objectives and is thus self-motivated.[2] The difference between $w$ and $c$ can thus be interpreted as a compensation for the restriction on the freedom of the developer, an agency cost reflecting informational asymmetry between the developers and the entrepreneur, the cost of protecting the software from unauthorized copying, or the cost to set up a firm and manage it. Part 6 discusses how $w$ and $c$ probably evolved in the last few decades, and how that evolution may have influenced the developers' license terms choices.

**Literature** Open source license are a variation on the copyright law. They dictate the authorized use of a piece of software by appealing to the legal protection the State grants to software authors under the form of a copyright. This paper thus belongs to the large literature on the systems of protection of the intellectual property, and more particularly to that strand which attempts to determine the optimal legal protection that the State should grant to inventors. Besen-Raskind (1991) and Menell (1999) are good general references.

Arrow (1962) discussed the trade-off between ex ante adequate pricing – which must allow the inventor to recoup the fixed costs of production of an information good – and ex post efficient pricing. Ex-post efficiency requires that a product be sold at marginal cost of production. That cost is near zero in the case of information goods such as software – reproduction and distribution costs are negligible on the Internet. In the absence of an intellectual property protection system, an invention can be copied freely and its price is soon driven to zero. Inventors are then not incited to release

[2]Note however that many developers work on OS projects under contract with a firm, and didn't actually choose to do so.

their invention to others, and may even decide not to incur the cost of producing that invention if its use to them is too low. Hirshleifer (1971) pointed out however that an inventor may be motivated to release her invention when its widespread diffusion brings private benefits. Beginning with Nordhaus (1969), the proper design of the intellectual property protection schemes has been discussed. That literature centers on the distinction between patents, licenses and trade secrets and discusses the duration and the scope of the protection they provide. There are however very few papers that go into the analysis of the interesting responses that the software developers' community put in place to solve the entrepreneur's Schumpeterian dilemma. That community crafted variations on the copyright laws that span all the way from strict protection of the invention – releasing it only under compiled form and having it developed 'in-house' by people bound by employment contracts that restrict their ability to make an own-use of the invention – to free release of the invention to the community of developers at large, requiring any improvements to be as free as the invention – GPL license terms. Bezroukov (2003) is among the few to have analyzed the difference between the various open-source software licenses from a social sciences point of view. Gaudeul-Crémer (2003) provides some elements for the analysis of the public policy aspects of the choice of open-source license terms vs. proprietary ones, while Lerner-Tirole (2002) is an empirical study showing when BSD-style license terms will be chosen instead of GPL ones. Lerner and Tirole show that the closer to market an open-source software is, the more likely it is that it will be GPL. In that paper however, commercial implementations of BSD-style software are not taken into account as those are not registered in the database used for the study – Sourceforge, a repository of open-source software. While there are a few papers attempting an analysis of the open-source license terms from an economic point of view (efficiency, determinants of the license terms' choices...), the present paper is also related to the literature about the protection of intellectual property. One of the important public policy objectives in that domain is the proper design of the legal apparatus around innovative industries. This paper shows that the GPL is bad for welfare when there is only one innovation and innovations do not build on each others. However, one of the main arguments for the GPL, which is that it leaves the 'innovations commons' open to all, is not examined here. Further work should therefore draw on the literature about sequential innovations.

Some phenomenon that were studied in the literature on R&D are present in this model; when a software is released under the BSD, there is a race to release a commercial implementation that is quite similar to a race to patent. Indeed, patents under the US system are granted based not only on who got the idea for development first, but also on whether the invention got actively developed. Grossman-Shapiro (1987) is a study of such race to patent, and it makes the same kind of distinction between the research and the development phase of an invention than in this model. They underline the potential waste of effort that is induced by that race to patent; a legal system that grants too much protection to an inventor can be welfare reducing, because too many people participate in the race for an uncertain outcome. When a software is released under the GPL, then another effect is present; since anyone's work will be freely released to others, developers may free ride on the efforts of others. This is what happens when the legal protection of intellectual property is too weak.

The debate over whether allowing generics or making drug patents available for free to third world countries would stifle innovation and reduce welfare, is one public policy problem that is similar to the debate over whether open-source software increases or decreases innovation (Scherer (1993)).

**Outline** The rest of the paper is organized in the following way: The first section presents the structure and primitives of the model. The second section compares the GPL and the proprietary license terms. They are relatively easy to analyze because once the initial developer chooses the license terms, others cannot change it. The third section introduces the BSD license terms, which can be changed to the GPL or to the proprietary license terms along its development and is thus more complex to analyze. The fourth section draws on the two previous ones to make a global analysis of the three license terms and shows when they will be chosen depending on the level of $w$ and $c$. The last section discusses the structure and primitives of the model to qualify its results, while providing some pointers for further study of the topic. The conclusion summarizes the main results of the model.

## 2 The model

One developer needs help on a project she has initiated. She will be called 'the project leader' or 'the innovator'. At that stage, she also has to set up an organization in order to publicize the software, attract developers, give a direction to its development, and protect her investment in the software (copyright). The choice of the license terms will determine in large part the character of the organization that will build around the software. Before seeking developers for her program, she has to choose what license terms will govern the development of her software. There are three types of license terms that can be used after having developed a program and when contemplating making it public: proprietary, GPL or BSD. There is an infinite pool of developers who can be called to work on a project. The project leader releases the program under compiled and/or source code form to one person at a time, chosen among a subset of users/developers and under specific usage restrictions. This means that in each period, there is only one new developer who is subjected to the source code of the project. There is only one possible idea for future development of the project over its whole life. Alternatively, once an idea has been generated, it becomes a focal point for other developers, who will not generate new ideas until that first idea is implemented. When that idea has been generated, either the one who generated it can implement it himself, or she will never be able to. This means there is no possibility for her, in a further period, to learn how to implement that idea if she was not able to implement it in the first place.

There is a probability $\lambda$ that an user/developer has an idea on where to lead the project. If an idea has been generated in one period, but not developed upon, then it can be transmitted next period to be developed upon. The decision to reveal the idea is under the control of the one who generated it.

There is a probability $\mu$ that an user/developer knows how to implement an improvement or innovation in the software. Anybody can with some probability $\mu$ implement an innovation proposed by somebody else, and even the one who generated an idea has the same probability $\mu$ only to know how to implement it.

This model therefore makes a clear distinction between the role of the user-developer-base as a source of market intelligence – knowing where to head the project – and as a source of developers – knowing how to implement an idea.

The project has no value, either to the leader of to others, if it doesn't get implemented. The project, if finished, has value $U$ to its leader and value $V$ in the market. Value $v$ is the value of the project to any individual user/developer other than the innovator.

**Assumption 1** $\lambda(V+v) > v$ *and* $v < U$

The fact that the software has no value if not developed upon by somebody else captures the collaborative nature of the development of the software: it is necessary, before the software be of any use to the innovator or to others, that the code be shown to somebody else developed upon by some people in the user/developer base. $U$ and $V$ capture the private and social value of the software, the first being the innovator's private motivations for developing a software and the second determining the motivation to ready it for the market. $U$ is strictly more than $v$ because you get more value from the successful completion of a software you shaped yourself and therefore fits your idiosyncratic needs than from a software that was initiated by somebody else. $\lambda(V+v) > v$ means that the market value of the software is sufficiently high to make of it a valuable project to work on. $U$, $v$ and $V$ can also be seen as the *increase* in the value of the software that the idea, if implemented, would generate. They do not have to be absolute values for the software.

The software does not provide the innovator or anybody any utility before this collaborative phase. If the project receives an improvement idea which is developed upon, then it provides the innovator with utility $U$ and the market with utility $V$. From the structure of the user/developer base, this happens only with probability $\lambda\mu$ in a first period. Indeed, the values $(U, V)$ can be obtained only if, either in one period, or in more, both competencies on the 'what' and the 'how' of the project have been put to work. The idea generation/development game lasts until the project is completed.

## 2.1 The proprietary model

The innovator has got different ways to contact the user/developer base: She can hire a developer and hope for her to have an idea for development. Hiring a developer costs $w$ ('wage'), but when hiring a developer, there is only probability $\lambda$ that she will have an idea for development: it is not possible to make sure the developer will have an idea for development, since this would require releasing the code and having her work on it before even having hired her, which she will not accept due to moral hazard from the innovator's part: a developer will not reveal an idea before being paid,

as the innovator would not pay her after she revealed it.[3] It is also not possible to make sure she would know how to develop on an idea if one exists, for the same kind of reason as above.[4] In this scenario which corresponds to the software having proprietary license terms, the original developer will acquire, in addition to $U$, the market value $V$ of her project. The wage $w$ paid to the developer must compensate the developer for her opportunity cost of working on the project. It is a market wage, as developers compete for work on projects and firms (innovators) compete for developers. This market for developers is not modeled here, but it results in a wage set at $w$ which is therefore the outside opportunity for a developer.

The per-period cost of working on the project is $c$ : it is incurred whether you get an idea for development or not, and is the same whether you are implementing an idea and/or generating one. $c$ is the cost to understand the program and trying to write some code. An idea cannot be generated without first working on the code, so that the cost $c$ is incurred even if no idea is generated. If there are not enough developers or equivalently too many projects, then $w > c$. It is possible to monitor that a developer incurs that cost $c$ if she is under contract and paid wage $w$. Alternately, a developer is motivated to generate an idea anyway, for the same kind of reason she was motivated to reveal it—see note 1.

A period is defined as a new round of disclosure of the program's source to a developer. The per-period discount factor for the project's value is $\delta$.

**Assumption 2** $\delta \geq 1 - \lambda$

## 2.2 The open-source models

If you do not want to follow the proprietary model, and therefore do not want to hire a developer, you can release the code in the public. Since the software has no value to anybody, as it does not work yet, you have to release the code and hope that an idea for its development will be generated, so as to get it implemented afterwards. Then of course, if somebody implements the idea, she can appropriate the software, unless the innovator chose the GPL, in which case nobody appropriates the improvement. The innovator cannot release the source code under proprietary license terms and hope for others to work on it for free, because the innovator would acquire all benefits from their contribution and cannot be trusted to leave anyone any surplus.

If the software is open-source ("BSD" license terms), then the innovator does not have to pay anybody for its development, but she also does not have any control on the use that is going to be made of it. The difference with the choice of making it free-source ("GPL" license terms) is that under the BSD license terms, a developer can change the software's license terms to proprietary

[3] Now, the question may be 'why would a developer reveal her idea when she is paid even if she doesn't have one?' Indeed, revealing an idea can be hard work, as it involves communicating with others and being intelligible... Assume there is a reputation game going on and a developer needs to be recognized to advance her career or to satisfy her ego. See Johnson (2003) for a model where employees in a proprietary project prefer not to reveal their ideas for development out of fear those will be adopted and they will be required to implement them.

[4] There are no different levels of ability in this model, or equivalently, no way to screen developers based on their ability level.

ones in a second period. The difference with releasing it under proprietary license terms and hiring a developer is that under proprietary license terms, any idea generated by a developer under contract is the property of the innovator.

Conversely, if the innovator release the software under the GPL, then she ensures that if it is developed upon by somebody else, she will be able to obtain this improvement for free, but that also means she cannot choose to make it closed source thereafter.[5] Also, the incentives for another developer to work on the program are limited to what she would personally gain from the successful completion of the project, $v$.

This graph shows the intra and inter-period revelation of ideas and developments, including the choice of whom to reveal the idea to.

(Graph 1 p. 11)

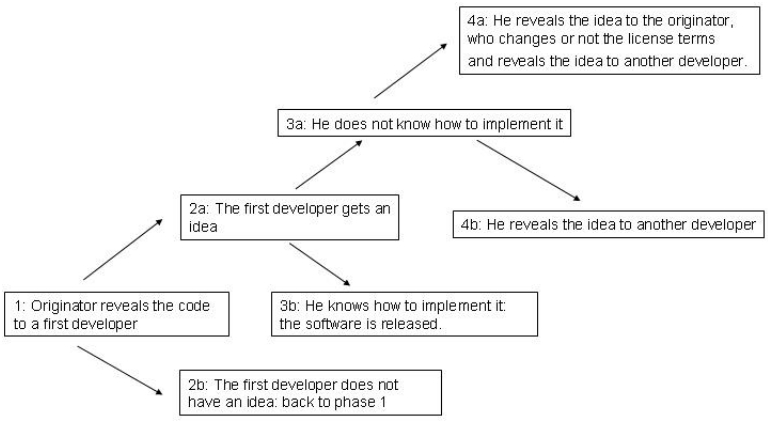


Figure 1: Intra and inter-period revelation of ideas.

There may be changes in the license terms for a project: a proprietary

project may become BSD, and vice-versa, but a change to a GPL license term is irreversible. Apart from the original developer of the project, nobody can change the license of a GPL software, and then, even the original developer can change the license only on the code she originally released. A developer cannot change the license terms of a software unless she, or somebody she hired, added some code which implements a meaningful idea, and the software was BSD to begin with. Even then, the change in the license terms affects only the part that was changed.

[5]There is in fact a provision in the GPL that reserves the copyright of the code writers if the code is used in ways that go against the GPL (e.g. making it proprietary). This is how the GPL is enforced. The project leader may also decide to change the license terms on the code she *initially* wrote, but cannot do so on the code that was added to it by others.

### 2.3 Some simplifying assumptions

There is no way to prove ownership of an idea unless it has been implemented—expressed into code writing and working. The only way to reveal an idea and still own its implementation is by paying the developer who implements it; she will sign a non-compete, no-disclosure contract.

The leader of the OS project releases the code because her cost in working on the program has risen; she will therefore not anymore work himself on the program; if a developer generates an idea and cannot develop on it, and even though the innovator could technically implement that idea, she will not do so because, for example, she has other more interesting projects to work on, or she got a full time programming job. This assumption is reasonable in view of the behavior of many OS projects leaders, who try to limit their involvement in the OS project they generated.

The developers community is tightly knit: The time when the developer first released the code openly is publicly known. When a developer is contacted by the innovator and asked whether she is ready to work on the program, she knows the identity of people who previously worked on the program and can contact them at no cost. The innovator can release the code to only one new developer per period, however, in that period, developers can contact each other. It is more costly to communicate and explain the code of a software to a new developer than to simply contact other developers who previously worked on the project and exchange information about it. The project's birth date is known because it will usually have been recorded when is was announced publicly, for example on Sourceforge.net, an OS projects repository. The developers' community is quite tightly knit because information about the contributions made by each developers is readily available, for example in CVS files, and there are not many developers working in a given software development area. The proofs of the propositions show how those assumptions influence the way information diffuses in the developers' community.

In a first part, the GPL and proprietary license terms will be examined and compared, then in a second part, the BSD license terms, which can revert to either one of those two previously studied license terms, will be studied. In a third part, the innovator's license terms choice will be defined as a function of the values of $w$ and $c$ and the special case of a competitive developers' market will be dealt with in a fourth part.

## 3 The GPL vs. the proprietary license terms

Proprietary license terms guarantee to the innovator that she will keep the ownership of her software, meaning that she will be able to make people pay to use it. However, this also means she will have to pay developers to work on its development. Therefore, as the required wage $w$ increases and $V$, the market value of the software, decreases, the proprietary license terms will become less attractive.

The GPL license terms mean that the innovator loses all rights to the future developments in her software, since any change to it falls under those same GPL license terms that discourage commercial exploitation of the software – it can be used for profit, or even sold, but the source

code must be supplied and copying it cannot be prohibited. Therefore, she loses the opportunity to make a profitable commercial exploitation of her software. This means neither she, nor anybody else, will want to pay a software programmer to work on her GPL-licensed software. Indeed, if the innovator found it worthwhile to pay a software programmer to work on this GPL licensed software, it would be even more worthwhile for the innovator to do so under proprietary license terms, since that would allow her to acquire the market value of the software for the same price $w$. As for another developer, she will find it even less worthwhile than the innovator to pay a programmer to work on the program since her private value for the software is lower than that of the innovator. This means a developer will have to be privately motivated to work on the software, so that her private cost $c$ of doing so must be sufficiently low in comparison with the private benefit $v$ she will get from using the completed software. It must also be low enough to justify working on the software now instead of simply telling the innovator to find somebody else to work on it[6] and wait for another developer to do the work in a next period.

### 3.1 The licenses' domains of definition

The proposition below shows in what $(w, c)$ domain GPL and proprietary license terms are 'feasible' – meaning they may generate positive welfare for the project leader. The GPL license terms are feasible for any level of $w$, as $w$ doesn't enter into account in the decision of a developer working on a GPL licensed software, while the proprietary license terms are feasible only as long as $w \leq \frac{\lambda\mu(U+V)}{1-(1-\lambda)(1-\mu)\delta}$ and $w \geq c$, i.e. the project has positive value and the wage must be incitative. However, a GPL licensed software will be worked on only for $c$ low enough to warrant working on the software for free.

**Proposition 1** *If the software is GPL, developers will work on it with some probability as long as* $c \leq \frac{\lambda\mu v}{1-(1-\lambda)(1-\mu)}$. *If the software is proprietary, it will be developed as long as* $c \leq w \leq \frac{\lambda\mu(U+V)}{1-(1-\lambda)(1-\mu)\delta}$.

**Proof.** The value of a proprietary software is easy to calculate: each period, the innovator hires a new developer until an idea is generated, and then she hires new ones until the idea is implemented. The payoff is $U + V$ for the innovator, as she gets the market value of the software, but she has to pay $w$ each period of development. The value of a GPLed software is more difficult to calculate: if an idea was generated, it is not sure a developer will want to implement it; the cost may be too high and she will prefer to wait for another developer to do it, doing some free-riding at the cost of some delay in the development. Only when $c$ is very low will she want with probability one to implement the idea. Knowing this, a developer who is asked to generate an idea will work on it with even lower probability for a given level of $c$ than a developer who is asked to implement an idea, as her work has lower probability to give rise to a rapid gain – an idea *and* an implementation must be found –

[6] Or, if she is a diplomat, telling her after a reasonable amount of time that she looked at the project in earnest and is very sorry not to have found anything to contribute...

than for a mere implementer. There are therefore many cases depending on $c$, and as $c$ increases, the value of the project to the innovator decreases. The graph below shows how the probability $y$ a developer will try to generate an idea for the software's development, and the probability $x$ a developer will work to implement such an idea, vary with $c$ when the software is GPL.

(Graph 2 p. 14)

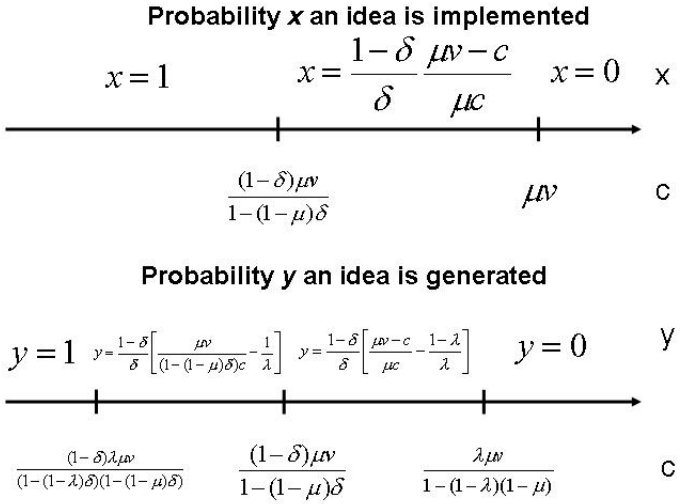


Figure 2: Probability an implementation or an idea is generated under the GPL.

**Notation 1** $c_1 = \frac{(1-\delta)\lambda\mu v}{(1-(1-\lambda)\delta)(1-(1-\mu)\delta)}$, $c_2 = \frac{(1-\delta)\mu v}{(1-(1-\mu)\delta)}$, $c_3 = \frac{\lambda\mu v}{1-(1-\lambda)(1-\mu)}$

Detailed calculations are shown in appendix $A$. ■

### 3.2 The comparison between the two licenses when $w = c$

The following graph shows how the value to the project leader of choosing the GPL license term varies with $c$, and also shows how this compares with the value of a proprietary license term when $w = c$. In this numerical example, the proprietary license term is preferred to the GPL for any level of $c$.

(Graph 3 p. 15) ($\lambda = \mu = \frac{1}{2}, \delta = \frac{8}{10}, v = 1, V = 9, U = 3$)

This can be generalized under some conditions:

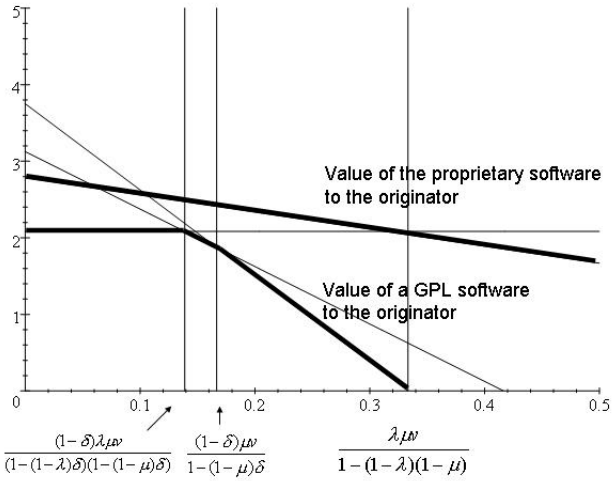


Figure 3: Value of the GPL and proprietary project to the innovator when $w = c$.

**Proposition 2** *The proprietary license terms will be preferred to the GPL when the developers' market is balanced, or $w = c$, as long as $V$ is sufficiently higher than $v$, and failing that, if $\delta$ is sufficiently high.*

**Proof.** See appendix $B$ where the condition is spelled out more precisely. ■

When $c$ is very low and $w = c$, the proprietary license terms are preferred to the GPL because the cost of hiring a developer is low too while proprietary license terms allows the developer to acquire most surplus from the software. Then as $c$ increases, the GPLed software is developed with ever lower probability which means that proprietary license terms will become even more advantageous. There are however cases where the value of a GPLed software doesn't diminish as fast as the value of the proprietary license terms, so that for medium level of $c$, the GPL may be preferred to the proprietary license term. This can be seen on the graph representing the variation in the value of the software as $c$ increases; even though for $c \leq c_1$ the proprietary license terms is always preferable to the GPL, we may see the value function under the proprietary license terms go below the value function under the GPL. This happens when it was not much higher the GPL one to begin with ($V$ small) and its slope is high compared to the slope of the GPL value function between $c_1$ and $c_2$ ($\delta$ is low). The influence of $\delta$ can be seen from the expression of the probability a software is worked on when it is under the GPL (see the figure 2); it is decreasing in $\delta$. That probability decreases as $\delta$ increases because the free-riding effect then plays to the full; developers do not have to wait long for another developer to come work on the project, and are therefore more inclined not to work on it now.

### 3.3 The comparison between the two licenses in the general case

This graph shows the comparison between the proprietary and GPL license

terms in the more general case when $w \geq c$.

(Graph 4 p. 16) ($\lambda = \mu = \frac{1}{2}, \delta = \frac{8}{10}, v = 1, V = 9, U = 3$)

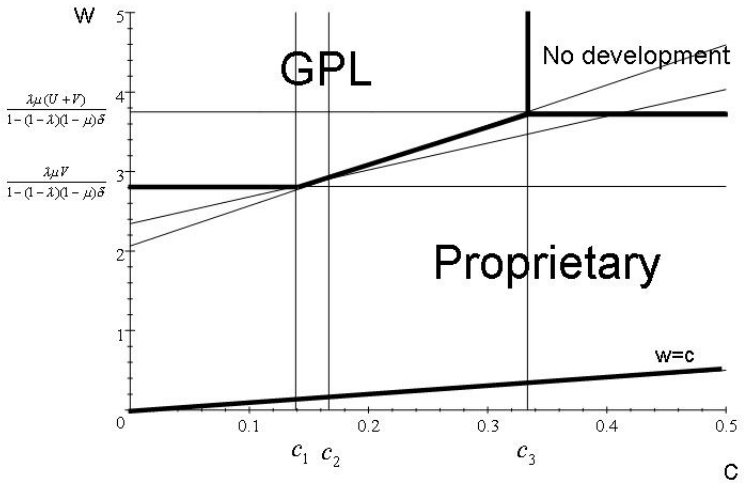


Figure 4: Choice of the GPL or proprietary license when $w$ and $c$ vary.

**Proposition 3** *As $c$ increases, the range of $w$ where the proprietary license terms are preferred increases, until they are the only ones feasible when $c \geq c_3$. As $w$ increases, the range of $c$ where the GPL is preferred increases until it is the only one feasible for $w \geq \frac{\lambda\mu(U+V)}{1-(1-\lambda)(1-\mu)\delta}$.*

**Proof.** Appendix $C$. ■

A developer, when choosing her license terms, will therefore have to look both at the cost of working on the project and at the conditions on the market for developers. If the costs are high but the market is balanced ($w$ not too much higher than $c$), she will choose proprietary license terms. She will use the GPL only when costs are low and the market is unbalanced.

### 3.4 A comparison of welfare

For $c \in [c_1, c_3]$ and $w \leq \frac{\lambda\mu(U+V)}{1-(1-\lambda)(1-\mu)\delta}$ (i.e. the proprietary license terms are feasible), the GPL may be chosen even though it is not optimal from a welfare point of view. Indeed, that choice leads to the software being developed with lower probability each period than if the proprietary license terms had been chosen. For any $w \geq \frac{\lambda\mu(U+V)}{1-(1-\lambda)(1-\mu)\delta}$, the GPL increases welfare because if that license didn't exist, the software would not be developed. Finally, in all other cases, the GPL achieves the same level of welfare than the proprietary license terms.

(Graph 5 p. 17) ($\lambda = \mu = \frac{1}{2}, \delta = \frac{8}{10}, v = 1, V = 9, U = 3$)

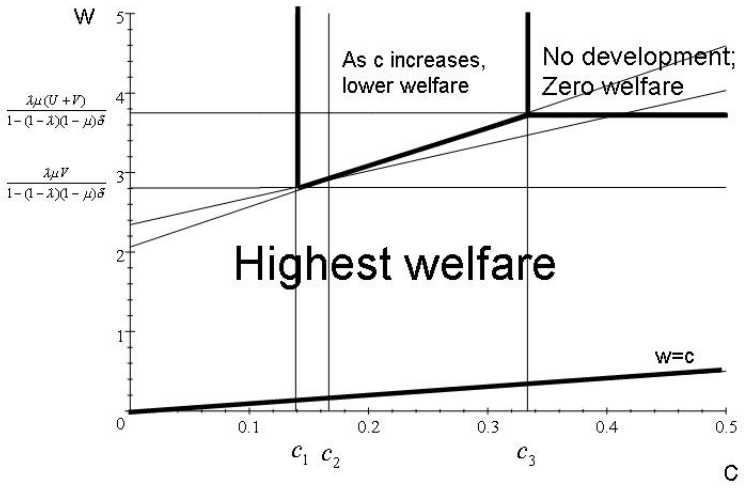


Figure 5: Welfare when the innovator has the choice between the proprietary and GPL license terms.

# 4 An analysis of the BSD license terms

When a software is BSD, and unlike under the GPL, a developer may change the license terms after having worked on the code which source was revealed to her. Unlike the proprietary license terms, the innovator does not keep the property of the improvements that were made by somebody else. Compared to the GPL, the BSD makes the software project more attractive to an outside developer because she may appropriate its market value.

The study of the BSD license terms is thus complicated by the possibility for any developer to change its license terms. In fact, a BSD software will ultimately always end up being either marketed under proprietary license terms if an outside developers implements an idea, or will see its license terms changed to the GPL once an idea has been generated.

The graph below shows the different cases that may occur when software is released under a BSD license. Some cases are easily seen to be impossible, and the cases that will be studied here are labeled BSD-BSD, BSD-Prop and BSD-GPL, which will be explained below.

(Graph 6 p. 18)

Some explanations for this graph are in order: When an user/developer implements an idea that was released under the BSD, she releases the implementation under proprietary license terms, whether that idea was her own or not, so as to capture its market value.

If she gets an idea but does not know how to develop on it, she can choose either to hire a developer to work on it under proprietary license terms, or release her idea to the innovator or another developer. When she releases her idea to the innovator, this one can make the software

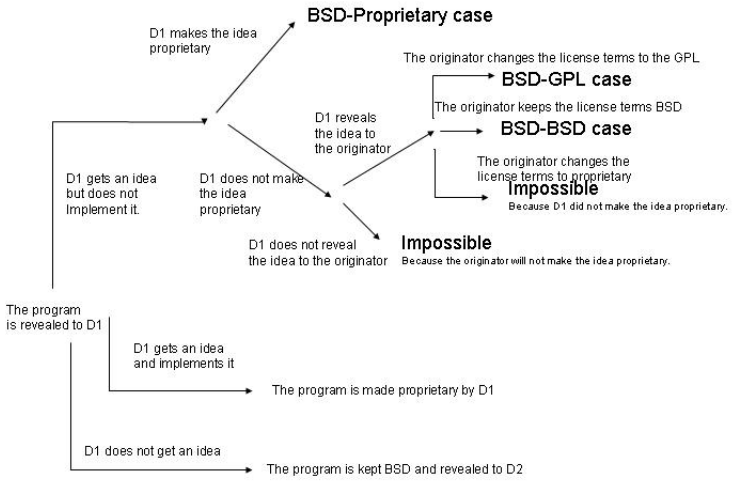


Figure 6: The BSD-BSD, BSD-GPL and BSD-Proprietary cases.

proprietary, keep the license terms BSD, or change them to the GPL. The innovator will not make it proprietary because then, the developer would have made it proprietary in the first place as both gain the same from making it proprietary. This also means the idea will be revealed to the innovator, since she will not change the license terms to proprietary ones.

The case where the first developer does not have an idea but hires a developer to work on the software can be discarded because $v < U$ : the original developer didn't choose to make her software proprietary in the first place, so that the first developer will not want to do so either.

This study of the BSD license terms can therefore be divided in three cases:

- First case: When an user/developer has an idea but doesn't know how to implement it, she reveals the idea to the innovator who keeps the license terms BSD and finds another developer to work on the idea. This is called the BSD-BSD case.
- Second case: When an user/developer has an idea but cannot develop on it, she hires a developer to code a proprietary software based on her idea. This is called the BSD-Proprietary case.
- Last case: When an user/developer has an idea but cannot develop on it, she reveals the idea to the innovator who then changes the license terms to the GPL. This is called the BSD-GPL case.

Once a developer has chosen the BSD license terms, then the different sub-cases will arise. The innovator will prefer the BSD-GPL sub-case, which allows her with some probability to get the program developed for free. Then if that is not possible, she will try to keep it under the BSD,

generating the BSD-BSD case. Finally, the worst case is when the license terms are changed to proprietary by the first developer who got an idea for development. Note that the BSD-GPL case would not be possible if the software was public domain; in the BSD, the innovator retains copyright and

can therefore change the license terms, while if the software was public domain, she couldn't do it. This is how the only reason why in this model 'BSD' and 'public domain' are not equivalent is that the GPL exists.

**Proposition 4** *Suppose* $w < \min[\mu(V+v), \frac{\lambda\mu(V+v)-c(1-(1-\mu)\delta)}{\lambda(1-\mu)\delta}]$. *Then various BSD-Proprietary cases arise, where from* $1$ *to an infinity of developers may potentially be contacted by the innovator to work on the project..*

*Suppose* $w > \mu(V+v)$. *Then if* $c < \mu v$, *the BSD-GPL cases arises, if* $\mu v < c < \lambda\mu(V+v)$ *the BSD-BSD case arises, and if* $c > \lambda\mu(V+v)$, *then the software is not developed.*

**Proof.** In appendix $D$. ■

The graph below illustrates the license terms choices of the developer:

(Graph 7 p. 19) ($\lambda = \mu = \frac{1}{2}, \delta = \frac{8}{10}, v = 1, V = 9, U = 3$)

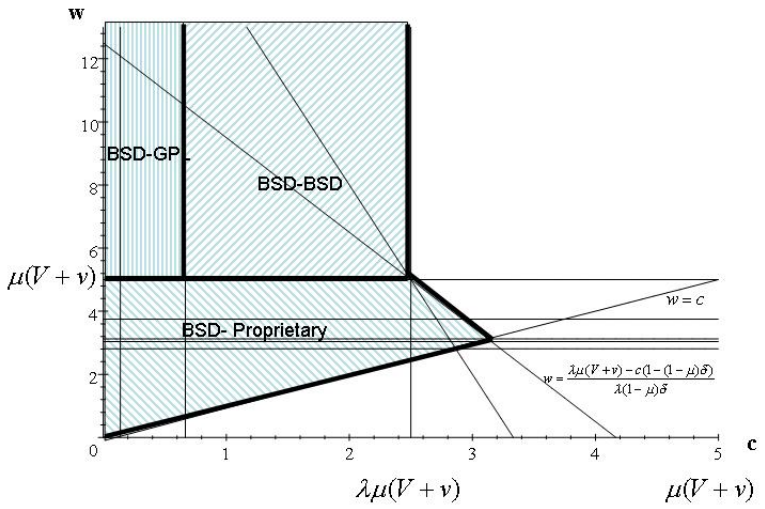


Figure 7: The different subcases arising out of the BSD when $w$ and $c$ vary.

The BSD-proprietary case is different from other cases: when a developer generates an idea, and since that idea is made proprietary, it is therefore not made public. When the innovator contacts a new developer in a next period, and even if an idea has been generated in a previous period, the development effort has to begin anew. While this may look very wasteful, this in fact generates

parallel development efforts and therefore increases the value of the project to the innovator. This effect is not present in any other cases, since those lead to an idea being either kept secret by the innovator himself, or made public. For $c \geq \lambda\mu(V+v)$, the welfare under the $BSD$ goes down. For example, in the case where only one developer comes to work on the project and there is therefore only probability $\lambda$ an idea is generated, then the welfare is very low.

The BSD-GPL case arises even when the BSD-BSD case would be more efficient from a welfare point of view, because the innovator prefers changing the license terms to the GPL and get value $U$ from it instead of keeping it BSD and get value $U - v$. This means that the change to the GPL will be decided even when it means there is a lower probability a developer may work to implement an idea that was generated in a previous period and not made proprietary. That inefficiency is higher the closer is $U$ to $v$.

(Graph 8 p. 20) ($\lambda = \mu = \frac{1}{2}, \delta = \frac{8}{10}, v = 1, V = 9, U = 3$)

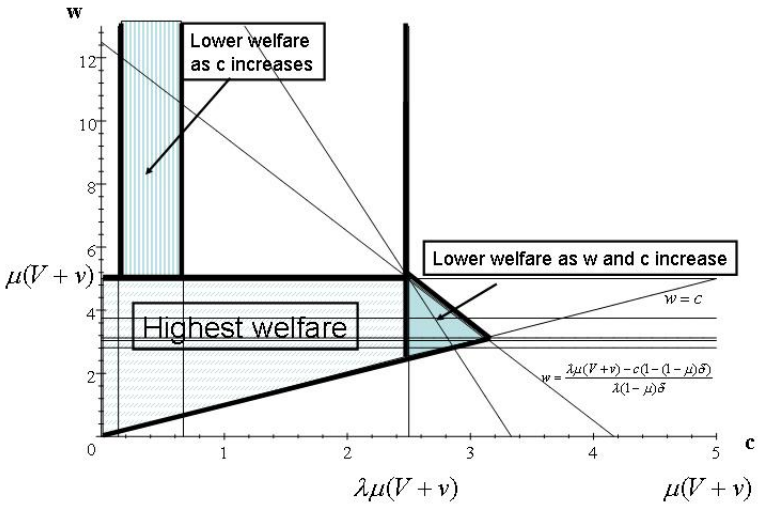


Figure 8: The variation of welfare as $w$ and $c$ vary under the BSD license terms.

# 5 The choice of the original developer

## 5.1 The general case

Now that both the GPL and proprietary license terms have been compared, and the BSD ones have been studied, it is possible to determine how the innovator will choose her software license terms when she has the three options available.

Before doing so, it is necessary to point out that neither a proprietary software or a GPLed one will see its license changed to the BSD. If the software is proprietary, it will not be changed to the

BSD (or the GPL for that matter) after an idea is generated. Indeed, that last pattern is obviously not reasonable, since the fact the idea was generated means there is more probability the software will finally get developed, and therefore paying a wage $w$ is even more justified. Even if an idea is not generated in the first few periods when the software is proprietary, the innovator's decision problem remains the same and she will keep the software proprietary. The same type of reasoning applies to GPLed software: a software that is GPL will be GPL forever.

The original developer when making her license terms choice for her project will take into account the end result in terms of personal welfare, but also the feasibility of each license terms choice; it makes sense to GPL a software only if $c$ is low, for example. When comparing the values of each choice to the innovator, the BSD-BSD case arises only for $c$ high and such that the GPL is not feasible. The comparison of the GPL with the BSD therefore needs to be done only in the BSD-GPL and BSD-proprietary subcases. The BSD-GPL may be preferred to the GPL when $c$ is high because then, the probability a developer works to generate an idea under the GPL becomes too low, while in the BSD-GPL case it is always one.

Additional effects enter into account when making the comparison between the GPL and BSD-Proprietary case: on the one hand, the project leader will have to pay for the software, but on the other hand, many developers will try to generate an idea each period, and the probability the software is developed is thus increased. The BSD when it leads to the BSD-Proprietary sub-case may be preferred to the GPL especially when $\delta$ is high, because in that case, the number of developers coming to work on the project rapidly increases, thus increasing quickly the probability that in each period an implementation is issued, and as seen before, free-riding effects then play fully to decrease the value of the GPLed software. When $U$ is a lot larger than $v$, so that having eventually to pay for the software is not very important, the BSD becomes preferable to the GPL.

The BSD license terms will always be preferred to the proprietary license terms when they lead to the BSD-BSD or BSD-GPL sub-cases. Indeed, those

cases arise only for $w$ very high, a domain where, even if feasible, the proprietary license terms are not interesting. However, when the BSD license terms lead to the BSD-Proprietary sub-cases, then the proprietary license terms may be preferred for $w$ sufficiently low. As $c$ increases, the $w$ domain where the proprietary licenses terms are preferred is widened, because the BSD license terms become more and more unfavorable for the innovator; there are less and less developers who come to work simultaneously on the project, so that its pace of development is slowed down.

It is now possible to determine the choice of the innovator according to the values of $w$ and of $c$ :

**Proposition 5** ***BSD vs. Proprietary:***

*When $w > \mu(V+v)$, the BSD license terms will be preferred to the proprietary ones.*

*For $w < \min[\mu(V+v), \frac{\lambda\mu(V+v)-c(1-(1-\mu)\delta)}{\lambda(1-\mu)\delta}]$, and $c < \lambda\mu(V+v)$, then the proprietary license terms will be preferred to the BSD for $w < \frac{\lambda\mu(V+v)}{(1-(1-\lambda)(1-\mu)\delta)} - \frac{\lambda\mu(1-\mu)^2\delta\lambda(U-v)}{(1-(1-\lambda)(1-\mu)\delta)(1-(1-\mu)(1-\lambda\mu)\delta)}$.*

*For $w < \min[\mu(V+v), \frac{\lambda\mu(V+v)-c(1-(1-\mu)\delta)}{\lambda(1-\mu)\delta}]$, and $c > \lambda\mu(V+v)$, that limit progressively increases as $c$ increases up until the proprietary license terms are preferred to the BSD for any*

$w < \frac{\lambda\mu(V+v)}{(1-(1-\lambda)(1-\mu)\delta)}$.

*Finally, when* $\mu(V+v) > w > \frac{\lambda\mu(V+v)-c(1-(1-\mu)\delta)}{\lambda(1-\mu)\delta}$ *the proprietary license terms are the only ones possible.*

***BSD vs. GPL***

*For* $w > \mu(V+v)$*, the GPL will be preferred to any other license terms as long as* $c \leq c_1$*, and there will then be a limit belonging to* $[c_1, c_2]$ *above which the BSD is preferred.*

*For* $w < \mu(V+v)$*, and* $c \leq c_1$*, the BSD license may be preferred to the GPL if* $\frac{U}{v} > \frac{[1-(1-\mu)(1-\lambda)\delta]}{(1-\mu)^2\delta\lambda}$.

*The comparison of the GPL with the proprietary license term was done in proposition* $3$.

**Proof.** The comparison between the BSD and the proprietary license terms is easy

when the BSD leads to the BSD-GPL or BSD-BSD subcases. Indeed, those subcases arise only when $w > \mu(V+v)$ and this means the proprietary license is not feasible since it has positive value only for $w \leq \frac{\lambda\mu(V+v)}{1-(1-\lambda)(1-\mu)\delta}$ which is less than $\mu(V+v)$. Therefore, the only comparison to be made is between the GPL and the BSD-GPL and BSD-BSD subcases. However, the BSD-BSD subcases arises only for $c \geq \mu v > c_3$, a domain in which the GPL is not feasible. For $c \leq c_1$, the GPL is always preferred, since it leads to the same probabilities than the BSD that the software will be developed, and prevents the appropriation of an idea by a developer. Therefore, the two must be compared only for $c \in [c_1, c_2]$. See appendix $E$.

Since the comparison between the $GPL$ and the proprietary license terms was done in proposition $3$, there remains only to compare the $GPL$ with the BSD in those cases where the BSD is preferred to the proprietary license terms, or when $w < \frac{\lambda\mu(V+v)}{(1-(1-\lambda)(1-\mu)\delta)} - \frac{\lambda\mu(1-\mu)^2\delta\lambda(U-v)}{(1-(1-\lambda)(1-\mu)\delta)(1-(1-\mu)(1-\lambda\mu)\delta)}$. See appendix $E$. ■

The graph below illustrates the comparison and outlines the regions where the BSD, the GPL or the proprietary license terms will be used.

(Graph 9 p. 23) ($\lambda = \mu = \frac{1}{2}, \delta = \frac{8}{10}, v = 1, V = 9, U = 3$)

## 5.2 A competitive developers' market: $w = c$

**Proposition 6** *When the developers market is balanced, no software will be released open-source, except for* $c$ *medium and* $V$ *and* $\delta$ *low where the GPL may be preferred to the proprietary license terms (Proposition* $2$*). Even though some software projects may be launched under the BSD software license, their implementations will all be released to the public under proprietary license terms.*

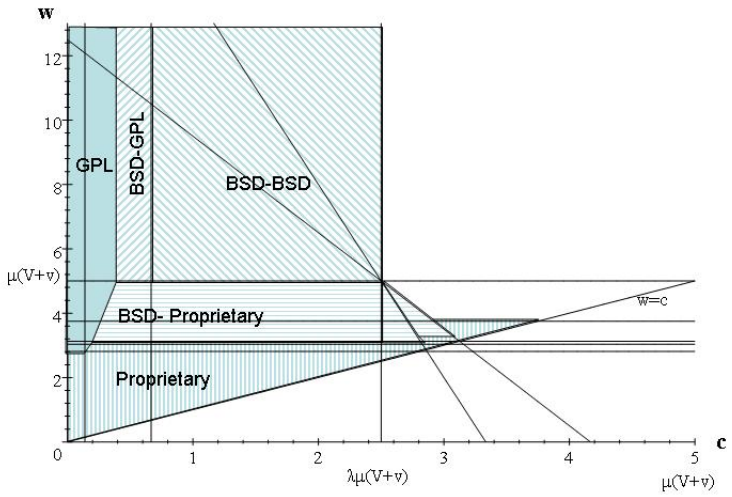


Figure 9: The choice between the GPL, the BSD and the proprietary license terms when $w$ and $c$ vary.

**Proof.** The BSD-BSD and BSD-GPL subcases arise in conditions that make it impossible that $w = c$, and therefore, when $w = c$, the license terms that must be compared are the BSD-Proprietary and GPL ones. The comparison with the GPL was done in proposition 2. The comparison with the BSD-proprietary needs only be done when that subcase brings the highest value for the leader, i.e. when $c \leq \lambda\mu(V + v)$. Indeed, if the BSD is not preferred in that case, it will never be preferred. See appendix $F$ for the condition under which the BSD is preferred. Even if it is preferred, it still results in the software being released under a proprietary license. ■

The BSD license will not lead to a sharing of ideas between developers when $w = c$, indeed, while an innovator may share her code source for others to improve upon, developers will keep their ideas to themselves and get them developed in-house by developers under contract.

### 5.3 A comparison of welfare

The existence of the BSD license terms unambiguously increases the welfare when $c \leq \lambda\mu(V + v)$. Indeed, either it is used when neither the proprietary nor the GPL license terms are feasible (this is notably the case when the BSD results in the BSD-BSD subcase), or, when the GPL and the BSD are both feasible and the BSD is preferred, it results in a higher probability the software is developed, and finally, when the BSD results in the BSD-Proprietary subcase, it also results in the software becoming developed with higher probability each period than in any case because more developers come to work simultaneously on it.

Mathematically, as long as $c < \lambda\mu(V+v)$, in which case an infinity of developers can be brought to work on the project, then the social welfare generated by the BSD is always higher than the welfare generated by other license terms:

$$\frac{\lambda\mu(U+V) - c + \lambda(1-\mu)\delta[\frac{\mu(U+V)-c}{1-(1-\mu)\delta}]}{1-(1-\lambda)\delta} <$$
$$\frac{\lambda\mu(U+V) - c + \frac{\lambda(1-\mu)\delta(U+V-\frac{c}{\mu})}{1-(1-\mu)\delta} - \frac{\lambda(1-\mu)\delta[(1-\mu)(1-\lambda\mu)(U+V-\frac{c}{\mu}]}{1-(1-\mu)\delta(1-\lambda\mu)}}{1-(1-\lambda)\delta} = SW_{BSD-\mathrm{Pr}\,op}$$

because

$$1 - \lambda\mu < 1$$

In this model, the BSD license, when it results in the BSD–Proprietary case, generates the highest welfare, as it allows for many developers to work simultaneously on the project, and therefore accelerates its completion. However, the assumption that each developer in each period knows when to invest $c$ in the development of the project is not very realistic, and in reality, there can be a waste in development efforts, that may counter the benefit from having more developers work on the project.

For $c \geq \lambda\mu(V+v)$, the $BSD$ may reduce welfare: in the case where only one developer comes to work on the project and there is therefore only probability $\lambda$ an idea is generated, then the welfare is unambiguously lower than in the proprietary case, where a developer will work on the project in each period until an idea is generated. However, as soon as for example 2 developers come to work on the project, the BSD may overtake the proprietary license in terms of welfare due to the competition

between developers. The advantage of the BSD-Proprietary case diminishes as $c$ increases, as less developers are brought to work on the project.

BSD-GPL cases would generate lower welfare, and be chosen less often by the innovator.

# 6 Discussion

The open-source license terms are the result of an imbalance in the software developers' market, and wages that are inflated over their competitive level. This has clearly been the case in the profession in past years, when software companies were constantly facing a shortage in software developers. A study could also be done comparing the level of qualification and expected wage of developers in the open-source community and those working in software companies. A link

could also be made between the move of software development centers for software companies toward third-world countries (particularly India) and the fact that open-source software developers predominantly come from developed countries with severe shortage of developers. The USA for example put in place H1-B visa programs to allow for a higher quota of people with specific qualifications, notably engineers, software developers and programmers. Germany also encourages the immigration of programmers, creating specific legislation for the creation of a new category of work permits for engineers, programmers and other information technology specialists. This bears testimony to the tensions in the market for developers in those two countries that count leaders in the field (Microsoft, Oracle, Adobe in the US, SAP in Germany). Open-source license terms could be a response to imbalances in the software developer market in developed countries, forcing companies to relocate to countries where software developers have less outside opportunities, while developers in developed countries find themselves forced to use open-source license to see their software developed, because they do not have access to that third world countries developers' potential. Concurrent with a possible rise in the cost of hiring developers was a decrease in the cost of coordinating software projects over the Internet in an open-source fashion. Tools appeared to make such collaboration easier outside from a firm-setting, while the developers community also came to acquire common programming languages and developed a culture of collaboration with agreed-on rules of behavior. In the terms of this model, the difference between $w$ and $c$ had a tendency to increase.

In the history of the emergence of open-source license terms, permissive BSD-type license terms were the norm in university institutions up until the 80s, and more restrictive GPL license terms appeared in the 90s as there were more tensions in the developers market. This can be interpreted as a reaction by developers who were not anymore able to develop their software in-house due to the inflation in developers' wage; the only way for them to draw contributions was to release the software open-source, but they needed more restrictive license terms to protect their invention. It would be possible to establish a correlation between developers' wages and the relative frequency with which projects were released open-source. In the case of TEX, the first few years saw the release of many competing commercial projects, while the license terms were later codified and made increasingly restrictive into the 90s. It will be interesting to see if the open-source phenomenon will be impacted by the more difficult conditions in the developers' market now, where employment and wages do not rise as fast as they did during the Internet bubble years, and are pressured by the competition from developers in third world countries where big software companies now find an increasing portion of their workforce.

In a more general model, the software gets improved over time, with a function $f(x)$ being the value of an improvement to a project that has a value $x$. The project follows a value trajectory, from an initial stage where it has value $x$, to a second where it has value $f(x)$, a third where it has value $f(f(x))$, etc. The next step for the model is thus to introduce multiple periods of innovation, where the fact the software is GPL ensures that an innovation that could be a building step for others will be kept public. This is when the GPL may show its use. The GPL seems rather unfairly disadvantaged

compared to the BSD in this model also because it is possible for a developer to develop a GPL software and make a commercial implementation of it, for example by developing an user-interface or by selling a distribution of the software, *à la* Red Hat or Mandrake for Linux. This increases the motivations to work on the program. A disadvantage of the BSD that doesn't appear in this model is that since innovations are kept secret, there will be some anarchy in the development, with the possibility that two competing version of the same software appear. In that case, and if there is Bertrand competition, then the software has no more market value. Other extensions would allow for competing ideas, and revelation of the source code in a less controlled manner, to more people at the same time.

There are two effects that combine to the detriment of welfare in this model, and which have not been discussed before: developers cannot commit to deliver the software for free to the innovator, and the innovator cannot commit to keep the software license BSD. Those limits to the commitment power of the developers and of the innovator may be partially solved due to 'gentlemen's agreement' between developers, but developers certainly are conscious of the possibility their project may be held-up or a software that was BSD and therefore interesting to work on for profit may turn GPL later and thus be unexploitable. This is why the community has a strong interest in keeping grouped around the leader of the project and contribute back to her all improvements to the software – those who don't follow that norm get excluded, a phenomenon that is called 'forking' in the open-source community – so that the leader knows she does not face risks of expropriation and can keep the software BSD. Then, of course, there is one moment where the value of the software becomes too high and one developer breaks rank and decides not to give away the further improvements she made so as to make a proprietary exploitation of them. A multi-period model where the potential value of the software increases each period, while the force of social norms holding the project together are limited, would be interesting, as long as it is assumed the market is composed of people who do not have access to the open-source software – if not, then the multi-period game can be summed up in one period as each improvement adds the same increment to the value of the software each period, and therefore, as consumers compare the value of the OS program to the proprietary one, the increment in the market value of the software is what determines its price. It is also possible that there is one moment in the development of the software where it suddenly acquires market value, for example when all contributions begin to be put together and an interface is built, and that moment determines when it is going to be made proprietary.

# 7 Conclusion

The proprietary license terms are generally used when the developers' job market is balanced. The GPL's existence is always bad for welfare, while the BSD's existence may decrease welfare in some cases but increases it in all others. The GPL is bad for welfare when it is preferred to the BSD, because it reduces the incentives to develop the software while increasing the benefits of the innovators, so that her incentives go against the interest of the development of the software. While

the GPL does indeed allow development of a software when it is not possible under proprietary license terms, the BSD license terms achieve the same objectives and with higher probabilities. The BSD is particularly good for welfare as it encourages parallel developments, and in those cases, it is the existence of the proprietary license terms which may be bad for welfare. Overall, while it is possible to say that the GPL license terms are bad for welfare and the BSD is always better than the GPL for any combination of $c$ and $w$, the proprietary ones are bad for welfare when $c$ is low and good for $c$ high. Interestingly enough , the only reason in this model that the BSD license is not equivalent to the 'public domain' is that it allows to retain the copyright on the software and thus change its license to the GPL. This means that in a world without the GPL, then there would only be a choice between the proprietary license terms and putting the software in the public domain. Optimality in terms of welfare, which would mean prohibiting the use of GPL license terms, would also mean BSD license terms would in effect be no different from public domain terms. Open-source licenses are therefore superfluous according to this very limited model.

# References


[1] Arrow, K.J. (1962): "Economic Welfare and the Allocation of Resources for Inventions", in Nelson, R.(ed.), *The Rate and Direction of Inventive Activity*, New York, Princeton University Press.

[2] Besen S.M. and L.J. Raskind (1991): "An Introduction to the Law and Economics of Intellectual Property (in Symposium: Intellectual Property)", *The Journal of Economic Perspectives*, **5(1)**, pp. 3-27.

[3] Bezroukov N. (2003) "BSD vs. GPL: A Framework for the Social Analysis", http://www.softpanorama.org/Copyright/License_classification/index.shtml

[4] Biais B. and E. Perotti (2003): "Entrepreneurs and new ideas", *Working Paper*.

[5] Cowan R., N. Jouard and M. Özman (2003): "Knowledge dynamics in a network industry", *Working Paper*.

[6] Crémer J. and A. Gaudeul (2003): "L'économie du logiciel libre", GDR TICS CNRS, Carry le Rouet, 7–12 septembre 2003.

[7] Dalle J-M and N. Jullien (2001): "Open source vs Proprietary software", *Working Paper*.

[8] Gaudeul A. (2003): "The (La)TEX project: A case study of open-source software", *Working Paper*.

[9] Grossman G.M. and C. Shapiro (1987): "Dynamic R&D Competition", *The Economic Journal*, **97(386)**, pp. 372-387.

[10] Hirshleifer, J. (1971): "The Private and Social value of Information and the Reward to Innovative Activity", *American Economic Review*, **61)**, pp. 561-574.

[11] Johnson J.P. (2003): "Collaboration, peer review and open source software", *Working Paper*.

[12] Johnson J.P. (2002): "Open Source Software: Private Provision of a Public Good", *Journal of Economics & Management Strategy*, **11(4)**, pp. 637-662.

[13] Johnson J.P. (2000): "Some economics of open source software", http://opensource.mit.edu/online_papers.php

[14] Lerner J. and J. Tirole (2002): "The scope of open-source licensing", NBER Working Paper 9363.

[15] Menell P.S. (1999): "Intellectual Property: General Theories", Encyclopedia of Law and Economics, Edward Elgar and the University of Ghent Publishers, Bouckaert B. and De Geest G. Editors.

[16] Mustonen M.(2002): "Copyleft - the economics of Linux and other open source software", *Working Paper*.

[17] Nordhaus W.D. (1969): "Invention, growth and welfare: A theoretical treatment of technological change", Cambridge, MA, MIT Press.

[18] Scherer F.M. (1993): "Pricing, Profits, and Technological Progress in the Pharmaceutical Industry", *The Journal of Economic Perspectives*, **7(3)**, pp. 97-115.

[19] Schumpeter, J.A. (1934): "The theory of economic development", Cambridge, Mass., Harvard University Press

[20] Schumpeter, J.A. (1928): "The Instability of Capitalism", *Economic Journal*, 38(151), pp.361-386.

# A Proof of proposition 1

Naming Convention: The 'first developer', denoted D1, is the one who works on the project when an idea has not yet been generated. The 'second developer', denoted D2, is the one who works to implement an idea that was generated by another developer.

**When the project is proprietary,** the value to the developer of working on the project is $w - c$, while the value of the project to its leader is

$$W_0 = \frac{A}{1 - (1 - \lambda)\delta}$$

with

$$\begin{aligned} A &= \lambda\mu(U+V) - w + \lambda(1-\mu)\delta[\mu(U+V) - w + (1-\mu)\delta[\mu(U+V) - w + ...] \\ &= \lambda\mu(U+V) - w + \lambda(1-\mu)\delta[\frac{\mu(U+V)-w}{1-(1-\mu)\delta}] \\ &= \frac{\lambda\mu(U+V) - w(1-(1-\lambda)(1-\mu)\delta)}{1-(1-\mu)\delta} \end{aligned}$$

Indeed, if the first developer who is contacted has an idea and implements it, which has probability $\lambda\mu$, then this brings private utility $U$ to the innovator, and she can sell the software for a total value of $V$ in the market. The, if the first developer has got an idea, but does not implement it, which has probability $\lambda(1-\mu)$, then in a second period, the innovator hires another developer, who will implement the other's idea with probability $\mu$. If she doesn't, then a third developer is hired in a third period and so on. Each time a new developer is hired, the previous one leaves and is not paid anymore, since there is no possibility she may have an idea or be able to implement it if she wasn't able to do so in a previous period. Finally, if the developer doesn't have an idea, which has probability $1-\lambda$, then another is hired next period and the situation is the same as in the previous period. The game is repeated indefinitely because we assume the pool of users/developers is unbounded.

The proprietary license terms will be adopted only if the value to the innovator is more than $0$, or $\frac{\lambda\mu(U+V)}{(1-(1-\lambda)(1-\mu)\delta)} > w$.

The welfare generated is

$$SW = \frac{\lambda\mu(U+V) - c(1-(1-\lambda)(1-\mu)\delta)}{(1-(1-\mu)\delta)(1-(1-\lambda)\delta)}$$

**When the project is GPL,** a developer has an option not to work on a project when no idea has been generated yet, and that option has positive value because another developer may choose to work on the project in a next period and then, since the result will be kept GPL, she will not have to pay to use it. In the same manner, a developer may choose not to work on the implementation of an idea. . Denote $x$ the probability a developer works on the implementation of an idea, and $y$ the probability that a developer works to generate an idea. The value of the project to a developer working on it when no idea has been generated yet is: The value to a developer of working to implement an idea is

$$W_{D2} = \mu v - c + (1-\mu)\delta\frac{x\mu v}{1-(1-x\mu)\delta}$$

Indeed, in a first period, she implements the idea with probability $\mu$ and it has utility $v$ to her, while she incurs cost $c$ for her work. Then, with probability $1-\mu$ she is not successful and then, in a second period, another developer may choose to work with probability $x$ on the project, and will implement it with probability $\mu$. If this does not happen (probability $1-x\mu$) then the game is repeated next period.

The payoff of not working to implement an idea is

$$W'_{D2} = \delta \frac{x\mu v}{1-(1-x\mu)\delta}$$

so that $x$ will be chosen to make the developer indifferent between working and not working, or

$$W_{D2} = W'_{D2}$$

so that $x = 0$ for $c \geq \mu v$, $x = \frac{1-\delta}{\delta}\frac{\mu v - c}{\mu c}$ for $c \in [\frac{\mu v(1-\delta)}{1-(1-\mu)\delta}, \mu v]$ and $x = 1$ for $c \leq \frac{\mu v(1-\delta)}{1-(1-\mu)\delta}$.

Now, the value to a developer of working on a project to generate an idea is:

$$W_{D1} = A - c + (1-\lambda)[\delta y A + \delta^2(1-y\lambda)yA + ...]$$

with $A = \lambda\mu v + \lambda(1-\mu)\delta[x\mu v + \delta(1-x\mu)[x\mu v + ...]]$

Indeed, if the developer has an idea, (probability $\lambda$) she will implement it with probability $\mu$, and get value $v$ out of it, and if she

doesn't implement it, then she passes on the idea to another developer in the next period who will work on it with probability $x$ and implement it with probability $\mu$ or pass it on with probability $1 - x\mu$.

If she doesn't have an idea (probability $1-\lambda$) then in a next period, a developer will choose to work on the project with probability $y$ and has an idea, then the game is the same as in the first period. If not, then the game is put off by another period.[7]

If the developer decides not to work on the project, then the value of the project to her is:

$$W'_{D1} = \delta y A + \delta^2(1-y\lambda)yA + ...$$

Indeed, she doesn't spend $c$ in trying to develop the software, but on the other hand, she has to wait next period for another developer to work on the software and possibly implement it. Therefore, $y$ will be chosen to make the developer indifferent between working and not working, so that

$$W_{D1} = W'_{D1}$$

[7]Depending on whether the GPL license terms allow the innovator to change her license terms to proprietary ones, if a developer has an idea for development, she may choose not to reveal it to the innovator, preferring instead to reveal it to another developer. This is because while the innovator would be able to then change the license terms to a proprietary one, another developer cannot do so, so that revealing it to another developer ensures that if developed upon, the software will keep being free. However, a developer who has an idea doesn't have to reveal it to the innovator to prove she really had one, because there is no reason she would lie about it. She can simply tell the innovator she got an idea and will contact another developer to implement it. Even if the innovator could change the license terms to proprietary ones, there will therefore not be duplication of effort where the innovator contacts a new developer when the previous one already had an idea. This is because you can say you had an idea and be believed without having to reveal that idea.

and we will have

$$\frac{(1-\delta)(1-\delta+\delta x)\lambda\mu v}{(1-(1-y\lambda)\delta)(1-(1-x\mu)\delta)} = c$$

when $x$ and $y$ belong to the $]0,1[$ interval. There are three different values for $A$ depending on the value of $x$, an hence $c$, so that $y = 1$ for $c \leq c_1$, $y = \frac{1-\delta}{\delta}[\frac{\mu v}{(1-(1-\mu)\delta)c} - \frac{1}{\lambda}]$ for $c \in [c_1, c_2]$, $y = \frac{1-\delta}{\delta}[\frac{\mu v - c}{\mu c} - \frac{1-\lambda}{\lambda}]$ for $c \in [c_2, c_3]$ and $y = 0$ for $c \geq c_3$. $c_1 = \frac{(1-\delta)\lambda\mu v}{(1-(1-\lambda)\delta)(1-(1-\mu)\delta)}$, $c_2 = \frac{(1-\delta)\mu v}{(1-(1-\mu)\delta)}$, $c_3 = \frac{\lambda\mu v}{1-(1-\lambda)(1-\mu)}$. Check that $c_1 < c_2$ and $c_2 < c_3$ as $\delta \geq 1-\lambda$.

As the value of the project to the innovator is

$$W_O = \frac{y\lambda\mu U(1-\delta+x\delta)}{(1-(1-x\mu)\delta)(1-(1-y\lambda)\delta)}$$

then

$$W_O = \frac{\lambda\mu U}{(1-(1-\mu)\delta)(1-(1-\lambda)\delta)}$$

for $c \leq c_1$,

$$W_O = \frac{1}{\delta}[\frac{\mu U}{(1-(1-\mu)\delta)} - \frac{c}{\lambda}\frac{U}{v}]$$

for $c \in [c_1, c_2]$

$$W_O = \frac{1}{\delta}[U - \frac{c}{\lambda}\frac{U}{v}\frac{(1-(1-\lambda)(1-\mu))}{\mu}]$$

for $c \in [c_2, c_3]$

and

$$W_O = 0$$

for $c \geq c_3$.

# B Proof of proposition 2

The innovator will choose a license term that is feasible and then, among those feasible, the one which brings her the highest value. When both license terms are feasible, she compares the value of the project to her under both license terms.

From the previous proof, the GPL will therefore be preferred to the proprietary license terms for $c \in [0, c_1]$ when

$$\frac{\lambda\mu U}{(1-(1-\mu)\delta)(1-(1-\lambda)\delta)} > \frac{\lambda\mu(U+V) - w(1-(1-\lambda)(1-\mu)\delta)}{(1-(1-\mu)\delta)(1-(1-\lambda)\delta)}$$

or

$$w > \frac{\lambda\mu V}{1-(1-\lambda)(1-\mu)\delta}$$

If $w = c$, and since $c < \frac{(1-\delta)\lambda\mu v}{(1-(1-\lambda)\delta)(1-(1-\mu)\delta)} = c_1$, that condition can be verified only if $\frac{(1-\delta)\lambda\mu v}{(1-(1-\lambda)\delta)(1-(1-\mu)\delta)} > \frac{\lambda\mu V}{(1-(1-\lambda)(1-\mu)\delta)}$, which is not possible. Therefore, $w$ must me strictly more than $c$ for the GPL to possibly be preferred to a proprietary license when $c \leq c_1$.

Define $c_1 = \frac{(1-\delta)\lambda\mu v}{(1-(1-\lambda)\delta)(1-(1-\mu)\delta)}$, $c_2 = \frac{(1-\delta)\mu v}{(1-(1-\mu)\delta)}$,
$c_3 = \frac{\lambda\mu v}{1-(1-\lambda)(1-\mu)}$ the limits that determine the changes in the $y$ probabilities the software is developed (see the graph 2). The two innovator's value functions, the one when she chooses the GPL and the one when she chooses the proprietary license terms and $w = c$, are piecewise linear in $c$. The slope of the GPL's value function is lower between $c_1$ and $c_2$ than between $c_2$ and $c_3$, and the proprietary value function is always higher than the one under the GPL for $c$ between 0 and $c_1$ (as seen above) and reaches 0 after the GPL value function reaches 0 ($c_3$ is less than $\frac{\lambda\mu(U+V)}{1-(1-\lambda)(1-\mu)\delta}$), so that it will always be higher than the GPL if it is higher at $c_2$. This is true if

$$(1-\delta)v(1-(1-\lambda)(1-\mu)\delta) < (1-(1-\mu)\delta)\lambda[V + U\frac{(1-\delta)^2(1-\lambda)}{\lambda^2\delta}]$$

and this is always verified if $(1-\lambda-(1-\lambda)(1-\mu)\delta)(1+\frac{(1-\delta)^2}{\lambda\delta}) \geq (1-(1-\lambda)(1-\mu)\delta)(1-\delta)$ (setting $V$ and $U$ at their lowest, where $\lambda(V+v) = v$ and $U = v$.). In turn, this is verified if $\lambda \leq \delta\,[1-(1-\lambda)(1-\mu)\delta]$ or $\lambda \leq \delta\frac{1-(1-\mu)\delta}{1-(1-\mu)\delta^2}$

The condition above comes from the fact that when $c = c_2$, the proprietary license terms are preferred when:

$$\frac{1}{\delta}[\frac{\mu U}{(1-(1-\mu)\delta)} - \frac{c_2}{\lambda}\frac{U}{v}] > \frac{\lambda\mu(U+V) - c_2(1-(1-\lambda)(1-\mu)\delta)}{(1-(1-\mu)\delta)(1-(1-\lambda)\delta)}$$

which translates as

$$(1-\delta)v(1-(1-\lambda)(1-\mu)\delta) < (1-(1-\mu)\delta)\lambda[V + U\frac{(1-\delta)^2(1-\lambda)}{\lambda^2\delta}]$$

# C Proof of Proposition 3

For $c \leq c_1$, and $w > \frac{\lambda\mu V}{1-(1-\lambda)(1-\mu)\delta}$, the GPL is preferred. For $c \geq c_3$, the proprietary license terms are the only ones feasible.

There thus remains to compare the GPL with the proprietary license terms for $c \in [c_1, c_3]$.

For $c \in [c_1, c_2]$, the GPL is preferred for

$$\frac{1}{\delta}[\frac{\mu U}{(1-(1-\mu)\delta)} - \frac{c}{\lambda}\frac{U}{v}] > \frac{\lambda\mu(U+V) - w(1-(1-\lambda)(1-\mu)\delta)}{(1-(1-\mu)\delta)(1-(1-\lambda)\delta)}$$

or

$$\lambda\mu U(1-(1-\lambda)\delta) - \lambda\delta[\lambda\mu(U+V) - w(1-(1-\lambda)(1-\mu)\delta)] > c\frac{U(1-(1-\mu)\delta)(1-(1-\lambda)\delta)}{v}$$

so that as expected, as $w$ increases, the range of $c$ for which the GPL is preferred increases.

The same type of comparison can be made for $c \in [c_2, c_3]$ but are not very inspiring either.

# D Proof of Proposition 4

**BSD-Proprietary: An idea is made proprietary by a developer.**

This case will happen only if $w < \mu(V + v)$. Indeed, the developer who got an idea then finds it profitable to hire a developer to work on it. The idea will not be released publicly, so that a second developer may then be contacted by the innovator in the second period to try to generate an idea for the software; there may be concurrent development.

For simplicity, assume that if all developers up to the $n^{th}$ developer did not implement their ideas in this period, and a $(n+1)^{th}$ developer comes to work on the project in the next period, then, in that next period, she learns whether any of the preceding developers implemented their idea before incurring any cost in trying to generate an idea and/or implementing one she had. This assumption reflects the fact that in a given period, preceding developers may be quicker than a newcomer in implementing an idea they worked on for a long time. A developer knows the identity of all previous developers contacted by the innovator and can contact them to know if they had ideas. Since there is no incentive for a developer in lying about having had an idea, and since it is easy to prove you implemented one, there is no room for false reporting. Note also that there is no point in working on an implementation to compete with an already existing implementation: in this model, there is only one possible idea, and they can be copyrighted once they have been implemented.[8]

$c < \lambda\mu(V + v)$ : **no limits on the number of developers coming to work on the project**

If $c < \lambda\mu(V+v)$, then there is nothing that can stop a new developer from entering the development process. Indeed, she will spend $c$ to develop the software only if she learned that nobody previously

[8]There are in fact examples of programs that were rewritten just so as to avoid paying for using them. The writing is protected, not the innovation.

working on the software released an implementation, and then her immediate expected value is $\lambda\mu(V+v)$. In a next stage, and if $w < \mu(V+v)$, she will spend $w$ on the wage of a developer if preceding developers did not release an implementation in this stage. If $w > \mu(V+v)$, then she will simply not hire any developer under any case.

The value of the project to a developer is therefore at least

$$W_D = \lambda\mu(V+v) - c$$

Indeed, when $n$ developers have already been brought to work on the project, the probability to have to spend $w$ to have a developer implement one's idea is very low, so that the value of the project gets reduced to what you get in the first stage, when you decide to work on the project seeing that nobody in this period released a software yet.

When $w < \mu(V+v)$, then the value of the project to the innovator is

$$W_O = \frac{A}{1-(1-\lambda)\delta}$$

with

$$\begin{aligned}
A &= A_0 + \lambda(1-\mu)\delta A_1 + \lambda(1-\mu)^2\delta^2\{\lambda A_2 + (1-\lambda)A_1\} + \\
&\lambda(1-\mu)^3\delta^3\{\lambda^2 A_3 + 2\lambda(1-\lambda)A_2 + (1-\lambda)^2 A_1\} + ... \\
&= \lambda\mu(U-v) + \lambda(1-\mu)\delta\{(U-v)[1-(1-\lambda\mu)(1-\mu)]\} + \\
&\lambda(1-\mu)^2\delta^2\{(U-v)(1-(1-\lambda\mu)^2(1-\mu)\} + \\
&\lambda(1-\mu)^3\delta^3\{(U-v)(1-(1-\lambda\mu)^3(1-\mu)\} + ... \\
&= \lambda\mu(U-v) + \frac{\lambda(1-\mu)\delta(U-v)}{1-(1-\mu)\delta} - \frac{\lambda(1-\mu)^2\delta(U-v)(1-\lambda\mu)}{1-(1-\mu)\delta(1-\lambda\mu)} \\
&= \frac{\lambda\mu(U-v)}{1-(1-\mu)\delta} + \frac{\lambda\mu(1-\mu)^2\delta\lambda(U-v)}{[1-(1-\mu)(1-\lambda\mu)\delta](1-(1-\mu)\delta)}
\end{aligned}$$

with

$$\begin{aligned}
A_n &= \mu(U-v) + (1-\mu)\mu(U-v) + ... + (1-\mu)^{n-1}\mu(U-v) + (1-\mu)^n\lambda\mu(U-v) \\
&= (U-v)[1-(1-\lambda\mu)(1-\mu)^n]
\end{aligned}$$

for $n \geq 1$ and $A_0 = \lambda\mu(U-v)$.

Suppose an idea is generated in a first period. There is then probability $\mu$ that an implementation is released, and then it will be sold at price $v$, since this is the price users are ready to pay for the software, the utility it brings to them, and there is no possibility for market discrimination. Then, if no implementation was made, there is probability $\mu$ that the idea is implemented by the developer hired by the first developer to have had an idea. If that doesn't happen, then the new developer

contacted by the innovator may himself have an idea and release an implementation. In a third period, there a two cases, either both developers who got brought to work on the project generated an idea, or only one did. This explains the term with $\lambda A_2 + (1-\lambda)A_1$. The same reasoning is used for the next periods.

If an idea was not generated in the first period, then the same game begins in a second period anew.

The welfare generated is

$$SW = \frac{\lambda\mu(U+V) - c + \frac{\lambda(1-\mu)\delta(U+V-\frac{c}{\mu})}{1-(1-\mu)\delta} - \frac{\lambda(1-\mu)\delta[(1-\mu)(1-\lambda\mu)(U+V-\frac{c}{\mu}]}{1-(1-\mu)\delta(1-\lambda\mu)}}{1-(1-\lambda)\delta}$$

$c > \lambda\mu(V+v)$**, and** $\mu(V+v) > w$

Since $c > \lambda\mu(V+v)$, then a developer will choose to work on the project only for $w$ sufficiently low so as to justify the initial investment of $c$ in the project.

There will be different cases depending on the level of $c$ where either 1, 2, or more developers will accept to work on the project because they have sufficient chances of getting a return on their initial investment of $\lambda\mu(V+v) - c$ in a future period.

Let us look first at the case were only one developer will come to work on the project: This is when a developer knows the source code has been released more than one period, then she has no sufficient incentive to work on it, because the probability another developer is already trying to make a proprietary software based on it is too high, and therefore, the probability to acquire the market surplus too low.

The value of the project to a developer is then:

$$\begin{aligned} W_{D1} &= \lambda\mu(V+v) - c + \lambda(1-\mu)\delta[\mu(V+v) - w + (1-\mu)\delta[\mu(V+v) - w + ...]] \\ &= \frac{\lambda\mu(V+v) - (1-\mu)\delta\lambda\max(w,c)}{1-(1-\mu)\delta} - c \end{aligned}$$

Indeed, she will be the only one working on it if she had an idea, so that there is no possible competition.

This must be positive, or

$$\frac{\lambda\mu(V+v) - c(1-(1-\mu)\delta)}{(1-\mu)\delta\lambda} > w$$

There is not going to be a second developer working on the project if:

$$[\lambda\mu(V+v) - c + \lambda(1-\mu)\delta\frac{(1-\mu)(\mu(V+v)-w)}{1-(1-\mu)\delta}] < 0$$

Indeed, this is the value of the project to a developer who comes second. With probability $\lambda$ she has an idea, and if she cannot implement it, she will pay a programmer to implement it only if, in each period, the programmer who was hired by the other developer was not able to implement the idea.

This translates in

$$\frac{(1-\mu(1-\mu)\delta)\lambda\mu(V+v) - (1-(1-\mu)\delta)c}{\lambda(1-\mu)^2\delta} < w$$

The developers will decide to work on the projects as long as their value of doing so is positive, because if they do not, then they will have to pay $v$ for any implementation of the program by another developer, so that the alternative to developing is of zero value. This is different from the $GPL$ case where there was a free-riding problem.

The value to the innovator is

$$\begin{aligned} W_0 &= \lambda\mu(U-v) + \lambda(1-\mu)\delta[\mu(U-v) + (1-\mu)\delta[\mu(U-v) + ...]] \\ &= \frac{\lambda\mu(U-v)}{1-(1-\mu)\delta} \end{aligned}$$

Existence conditions:

$$\frac{\lambda\mu(V+v) - c(1-(1-\mu)\delta)}{\lambda(1-\mu)\delta} > w > \frac{(1-\mu(1-\mu)\delta)\lambda\mu(V+v) - c(1-(1-\mu)\delta)}{\lambda(1-\mu)^2\delta}$$

Graphically, it is possible to see how as $w$ and $c$ decrease, more and more developers are brought to work on the project and the value to the innovator increases from $\frac{\lambda\mu(U-v)}{(1-(1-\mu)\delta)(1-(1-\lambda)\delta)}$ to $\frac{\lambda\mu(U-v)}{(1-(1-\mu)\delta)(1-(1-\lambda)\delta)} + \frac{\lambda\mu(1-\mu)^2\delta\lambda(U-v)}{[1-(1-\mu)(1-\lambda\mu)\delta](1-(1-\mu)\delta)(1-(1-\lambda)\delta)}$

Finally, if $c > \lambda\mu(V+v)$, and $w > \mu(V+v)$ then developers will simply not work on the project.

## BSD-BSD: An implementation is made proprietary, but not an idea, and that idea is kept BSD.

The first developer, with probability $\lambda\mu$, gets an idea, implements it, and makes the result proprietary. Then, she gets utility $V+v$ from the project, as she can sell the software at price $v$ to all $n$ people interested in it, including the innovator, and retires utility $v$ from it.

Or with probability $\lambda(1-\mu)$, she gets an idea but does not implement it. She will not choose to hire a developer to implement if $w > \mu(V+v)$ as explained previously. Then, the innovator will not change the license terms to the GPL, if there is not enough incentive for a developer to work on it under those terms ($\mu v < c$, see next part) and will not change the license terms to proprietary ones either, as $w > \mu(V+v)$. She will keep the software BSD, and other developers will try to develop upon the idea.

Finally, with probability $1-\lambda$, the first developer gets no idea, and the project moves to a second period, identical to the first.

In all those cases, the first developer bears cost $c$ for working on the project, whether she generates an idea or not, so that her expected return on working on the project is

$$W_{D1} = \lambda\mu(V+v) - c$$

If she does not work on it, then she gets value $0$ from the software, since if it is developed, she will have to pay for it.

If an idea was generated in the first period and not developed upon, and since the license terms are not changed to the GPL ($\mu v < c$), then the idea is released to a second developer and is developed upon with probability $\mu$. It then generates profits $V+v$ for the second developer (she sells it on the market for total proceed of $V$ and it provides utility $v$ to her). If it is not developed upon, it generates no utility for the developer who worked on the project as she will have to pay $v$ when the software is eventually developed, for a product that has utility $v$ to her, so that the expected return of working on the project for a developer who works on an idea generated by another is

$$W_{D2} = \mu(V+v) - c$$

This is higher than the value of the project before an idea has been generated, and therefore does not put additional constraints.[9]

The value of the project to the innovator can be found by recursive means:

[9]A developers will still try to generate an idea instead of waiting for another to generate an idea, if we assume that a developer who did not generate an idea is left out of the further rounds of disclosure of ideas, i.e. the originator decides not to reveal her ideas that another developer might have had.

In the case where the idea is revealed to previous developers who worked on the project if neither the originator nor the developer who got the idea can develop on it, then the condition is that:

$$W_{D1} > \delta\lambda(1-\mu)W_{D2} + \frac{1}{2}[\delta\lambda(1-\mu)]^2 W_{D2} + ...$$

(Assuming that when $n$ developers have worked on the project, there is probability $\frac{1}{n}$ for an individual developer to be brought again to work on the project)

$$A = \mu(U-v) + (1-\mu)\delta[\mu(U-v) + (1-\mu)\delta[\mu(U-v) + (1-\mu)\delta[...]]$$
$$= \frac{\mu(U-v)}{1-(1-\mu)\delta}$$

is the value of the project if in the first period an idea is generated,

and the overall value of the project to the innovator is:

$$W_O = \lambda A + (1-\lambda)\delta[\lambda A + (1-\lambda)\delta[...]] = \frac{\lambda A}{1-(1-\lambda)\delta}$$

Existence conditions: $\mu v < c < \lambda\mu(V+v)$, and $w > \mu(V+v)$.

Welfare generated: $SW = \frac{\lambda\mu(U+V)-c+\lambda(1-\mu)\delta\frac{\mu(U+V)-c}{1-(1-\mu)\delta}}{1-(1-\lambda)\delta}$

**BSD-GPL: An implementation is made proprietary, but not an idea, and that idea is made GPL if not developed upon by the innovator.**

In this part, the innovator changes the license terms to the GPL once an idea has been generated and not made proprietary. As long as the license terms remain BSD, a new developer can be confident no other developer is working on the project at the same time, so that there is not even the need for some communication mechanism between developers. The change in license terms acts as a signal that an idea was generated. That idea will be released to a next developer if not developed upon by the one who generated it, since the license terms have been changed to the GPL and there is therefore no possibility of expropriation.

Denote $x$ and $y$ the probabilities a developers works to implement an idea and the probability she works to generate an idea respectively.

The value to a developer of working to generate an idea for the project is

$$W_{D1} = \lambda\mu(V+v) - c + A + (1-\lambda)\delta[yA + (1-y\lambda)\delta[yA + ...]]$$
$$= \lambda\mu(V+v) - c + A + (1-\lambda)\delta\frac{yA}{1-(1-y\lambda)\delta}$$

with

$$\begin{aligned} A &= \lambda(1-\mu)\delta[x\mu v + (1-x\mu)\delta[x\mu v + (1-x\mu)\delta[...]]] \\ &= \frac{\lambda(1-\mu)\delta x\mu v}{1-(1-x\mu)\delta} \end{aligned}$$

Indeed, after the first period of development, if an idea was generated and was not implemented, then another developer will implement it with probability x$\mu$. If there was no idea generated, then the payoffs are set back by one period and another developer comes to work on the project with probability $y$.

The developer will try to generate an idea if this gives her higher surplus than waiting for another to do so: That last behavior generates

$$\begin{aligned} W'_{D1} &= \delta yA + (1-y\lambda)\delta^2 yA + [(1-y\lambda)\delta]^2\delta yA + ... \\ &= \frac{\delta yA}{1-(1-y\lambda)\delta} \end{aligned}$$

so that the condition is

$$\begin{aligned} W_{D1} &> W'_{D1} \\ \lambda\mu(V+v) + \frac{(1-\delta)A}{1-(1-y\lambda)\delta} &> c \end{aligned}$$

This is true as long as $\lambda\mu(V+v) \leq c$ and the following shows how, if $\lambda\mu(V+v) \geq c$, then $x = 0$ so that the BSD license would not be chosen.

Indeed, the value of the project to a developer who works on the idea of another is

$$\begin{aligned} W_{D2} &= \mu v - c + (1-\mu)\delta[x\mu v + (1-x\mu)\delta(x\mu v + ...)] = \\ &\quad \mu v - c + (1-\mu)\delta\frac{x\mu v}{1-(1-x\mu)\delta}. \end{aligned}$$

If she rejects the offer to work on the idea, or accepts it and does not work on it, and leave the innovator to find another developer in the next period to work on the idea, then the value of the project is

$$W'_{D2} = \delta[x\mu v + (1-x\mu)\delta(x\mu v + ...)] = \frac{\delta x\mu v}{1-(1-x\mu)\delta}.$$

so that she will decide to work on it if

$$W_{D2} > W'_{D2}$$

or

$$\frac{(1-\delta)\mu v}{1-(1-x\mu)\delta} > c$$

$x$ will be chosen to make her indifferent between working and not working so that for $c \geq \mu v$, $x = 0$, for $c \in [\frac{(1-\delta)\mu v}{1-(1-\mu)\delta}, \mu v]$, $x = \frac{1}{\mu}\frac{1-\delta}{\delta}(\frac{\mu v}{c} - 1)$ and for $c \leq \frac{(1-\delta)\mu v}{1-(1-\mu)\delta}$, $x = 1$.

Then the value to the innovator is

$$\begin{aligned} W_0 &= B + (1-\lambda)\delta B + [(1-\lambda)\delta]^2 B + ... \\ &= \frac{B}{1-(1-\lambda)\delta} \end{aligned}$$

with

$$B = \lambda\mu(U-v) + \lambda(1-\mu)\delta\frac{x\mu U}{1-(1-x\mu)\delta}$$

so that

$$W_0 = \frac{\lambda\mu(U-v)}{1-(1-\lambda)\delta} + \lambda(1-\mu)\delta\frac{\mu U}{(1-(1-\mu)\delta)(1-(1-\lambda)\delta)}$$

for $c \leq \frac{(1-\delta)\mu v}{1-(1-\mu)\delta}$,

$$W_0 = \frac{\lambda\mu(U-v)}{1-(1-\lambda)\delta} + \lambda(1-\mu)(1-\frac{c}{\mu v})U$$

for $c \in [\frac{(1-\delta)\mu v}{1-(1-\mu)\delta}, \mu v]$ and

$$W_0 = \frac{\lambda\mu(U-v)}{1-(1-\lambda)\delta}$$

for $c \geq \mu v$.

Existence conditions: $w > \mu(V+v)$ else, the developer who has an idea will make it proprietary. This also guarantees that the innovator will decide not to change the license terms to proprietary ones one an idea has been generated and revealed to her.

The second developer must work on the idea with some probability, or $\mu v > c$. The license will be changed to the GPL instead of being kept BSD once an idea has been generated and not implemented by its generator when the probability it is developed under the GPL times the value to the innovator

then ($U$) is more than the probability it is developed under the BSD times the value to the innovator in that case ($U - v$). This translates in the following condition:

$$U * x_{GPL} \geq (U-v) * x_{BSD}$$

Therefore, for $c \in [\frac{(1-\delta)\mu v}{1-(1-\mu)\delta}, \mu v]$, the condition is that

$$U * \frac{1}{\mu}\frac{1-\delta}{\delta}(\frac{\mu v}{c} - 1) \geq (U - v)$$

or

$$\frac{(1-\delta)\mu v}{(1-(1-\mu)\delta) - \frac{v}{U}\mu\delta} \geq c$$

while for any $c$ outside that range, the innovator prefers to change the license terms to the GPL.

**BSD-proprietary 2: An idea that is not implemented by the first developer is made proprietary by the innovator.**

Clearly, this case cannot happen when $w > \mu(V + v)$, and neither in the case where $w < \mu(V + v)$ and $c < \lambda\mu(V + v)$. In the case where $\max(w, c) < \mu(V + v)$ and $c > \frac{\lambda\mu(V+v)-(1-\mu)\delta\lambda w}{1-(1-\mu)\delta}$, you cannot have $c < \lambda\mu(V + v)$, so that the first developer will not work on the project. If $\max(w, c) < \mu(V + v)$ and $c < \frac{\lambda\mu(V+v)-(1-\mu)\delta\lambda w}{1-(1-\mu)\delta}$, then the first developer makes the software proprietary.

# E Proof of proposition 5

Since the condition for the existence of GPL implies that the value to the innovator of the BSD-Proprietary sub-case is the one where all developers potentially come to work on the project, GPL will be preferred to BSD-Proprietary for $c \leq c_1$ when

$$\frac{\lambda\mu U}{(1-(1-\mu)\delta)(1-(1-\lambda)\delta)} > \frac{\lambda\mu(U-v)}{(1-(1-\mu)\delta)(1-(1-\lambda)\delta)} + \frac{\lambda\mu(1-\mu)^2\delta\lambda(U-v)}{[1-(1-\mu)(1-\lambda\mu)\delta](1-(1-\mu)\delta)(1-(1-\lambda)\delta)}$$

or

$$\frac{[1-(1-\mu)(1-\lambda)\delta]}{(1-\mu)^2\delta\lambda} > \frac{U}{v}$$

which is the case as long as

$$\delta < \frac{1}{(1-\mu)(2-\mu-\lambda)}$$

($\mu$ and/or $\lambda$ low) and $U$ is not much higher than $v$.

Now, there remains to compare the proprietary license terms with the various BSD sub-cases. When $\frac{\lambda\mu(U+V)}{(1-(1-\lambda)(1-\mu)\delta)} > \mu(V+v)$, i.e. the proprietary license terms are feasible and the BSD-GPL and BSD-BSD sub-cases may occur, there is a comparison to be made between Proprietary and all BSD sub-cases, but when $\frac{\lambda\mu(U+V)}{(1-(1-\lambda)(1-\mu)\delta)} < \mu(V+v)$, then the comparison needs only be made with the BSD-Proprietary sub-case.

$\frac{\lambda\mu(U+V)}{(1-(1-\lambda)(1-\mu)\delta)} < \mu(V+v)$ **:**

Then the only comparison to be made is between BSD-Proprietary and Proprietary.

If $c < \lambda\mu(V+v)$, then the value of the BSD-Proprietary sub-case is maximal for the innovator, and it will be preferred to Proprietary when:

$$\frac{\lambda\mu(U-v)}{(1-(1-\mu)\delta)(1-(1-\lambda)\delta)} + \frac{\lambda\mu(1-\mu)^2\delta\lambda(U-v)}{[1-(1-\mu)(1-\lambda\mu)\delta](1-(1-\mu)\delta)(1-(1-\lambda)\delta)} > \frac{\lambda\mu(U+V)-w(1-(1-\lambda)(1-\mu)\delta)}{(1-(1-\mu)\delta)(1-(1-\lambda)\delta)}$$

or

$$w > \frac{\lambda\mu(V+v)}{(1-(1-\lambda)(1-\mu)\delta)} - \frac{\lambda\mu(1-\mu)^2\delta\lambda(U-v)}{(1-(1-\lambda)(1-\mu)\delta)(1-(1-\mu)(1-\lambda\mu)\delta)}$$

which is less than $\frac{\lambda\mu(U+V)}{(1-(1-\lambda)(1-\mu)\delta)}$, meaning that BSD-Proprietary is not always preferred to Proprietary.

When $c > \lambda\mu(V+v)$, and in the worst case for BSD-Proprietary, where only one developer comes to work on the project), it will be preferred to Proprietary when:

$$\frac{\lambda\mu(U-v)}{(1-(1-\mu)\delta)(1-(1-\lambda)\delta)} > \frac{\lambda\mu(U+V)-w(1-(1-\lambda)(1-\mu)\delta)}{(1-(1-\mu)\delta)(1-(1-\lambda)\delta)}$$

or

$$w > \frac{\lambda\mu(V+v)}{(1-(1-\lambda)(1-\mu)\delta)}$$

which of course is more than $\frac{\lambda\mu(V+v)}{(1-(1-\lambda)(1-\mu)\delta)} - \frac{\lambda\mu(1-\mu)^2\delta\lambda(U-v)}{(1-(1-\lambda)(1-\mu)\delta)(1-(1-\mu)(1-\lambda\mu)\delta)}$. Indeed, the value of BSD-Proprietary is lowest in this case, so that there is a narrower range where it is preferred to the proprietary license terms.

All the other cases, with more developers coming to work on the project, fall in between: the limit for $w$ is between $\frac{\lambda\mu(V+v)}{(1-(1-\lambda)(1-\mu)\delta)}$ and $\frac{\lambda\mu(V+v)}{(1-(1-\lambda)(1-\mu)\delta)} - \frac{\lambda\mu(1-\mu)^2\delta\lambda(U-v)}{(1-(1-\lambda)(1-\mu)\delta)(1-(1-\mu)(1-\lambda\mu)\delta)}$.

$\frac{\lambda\mu(U+V)}{(1-(1-\lambda)(1-\mu)\delta)} > \mu(V+v)$ **:**

Then, a comparison must be made between BSD-GPL and Proprietary and BSD-BSD and Proprietary:

Proprietary license terms will be preferred to the BSD-GPL when

$$\frac{\lambda\mu U}{(1-(1-\mu)\delta)(1-(1-\lambda)\delta)} - \frac{\lambda\mu v}{(1-(1-\lambda)\delta)} < \frac{\lambda\mu(U+V) - w(1-(1-\lambda)(1-\mu)\delta)}{(1-(1-\mu)\delta)(1-(1-\lambda)\delta)}$$

or

$$\frac{\lambda\mu(V+v) - \lambda\mu v(1-\mu)\delta}{1-(1-\lambda)(1-\mu)\delta} > w$$

but $\frac{\lambda\mu(V+v)-\lambda\mu v(1-\mu)\delta}{1-(1-\lambda)(1-\mu)\delta} < \mu(V+v)$ so that the BSD-GPL is always preferred to the proprietary license terms when both are possible.

Proprietary license terms will be preferred to the BSD-BSD when

$$\frac{\lambda\mu(U-v)}{(1-(1-\mu)\delta)(1-(1-\lambda)\delta)} < \frac{\lambda\mu(U+V) - w(1-(1-\lambda)(1-\mu)\delta)}{(1-(1-\mu)\delta)(1-(1-\lambda)\delta)}$$

or

$$w < \frac{\lambda\mu(V+v)}{(1-(1-\lambda)(1-\mu)\delta)}$$

which is also less than $\mu(V+v)$ so that BSD-BSD is always preferred to proprietary license terms when both are possible.

Those cases are not represented in the graph, since the numerical example that was chosen means that the proprietary license terms are not feasible when the BSD-GPL or BSD-BSD subcases arise.

# F Proof of proposition 6

Proposition 2 already stated under what condition the GPL license terms would be used in a developers' market in equilibrium. When the BSD license terms leads to the BSD-BSD or BSD-GPL sub-cases, it is not possible that $w = c$ because those sub-cases arise only when $w > \mu(V + v)$ and $c < \lambda\mu(V + v)$ and those two conditions are incompatible when $w = c$.

BSD-Proprietary license terms have the highest value to the innovator when $c < \lambda\mu(V + v)$ as explained before, and in that case, they may be preferred to proprietary license terms when $w = c$ if

$$\frac{\lambda\mu(V+v)}{(1-(1-\lambda)(1-\mu)\delta)} - \frac{\lambda\mu(1-\mu)^2\delta\lambda(U-v)}{(1-(1-\lambda)(1-\mu)\delta)(1-(1-\mu)(1-\lambda\mu)\delta)} < \lambda\mu(V+v)$$

Indeed, the left hand-side is the limit on $w$ over which the BSD-proprietary license terms are preferred to the proprietary ones, and this limit must be less than $\lambda\mu(V + v)$ otherwise it is beyond the $w \leq c$ domain.

This will happen when

$$\frac{(1-\lambda)(1-(1-\mu)(1-\lambda\mu)\delta))}{(1-\mu)\lambda} < \frac{U-v}{V+v}$$

which is possible for $U$ sufficiently large. Then, the loss in value from not acquiring the surplus from the market is not big enough to compensate for the savings in developers' wages that BSD-Proprietary allows.