Paper prepared for the 8th Annual CTI Conference

An Economic Perspective on Software Licenses – Incentives in Open Source Software

By Kasper Edwards[*]

Department of Manufacturing Engineering and Management, Technical University of Denmark
Building 423, office 028, 2800 Lyngby, Denmark. (email: ke@ipl.dtu.dk)

ABSTRACT

A reasonable explanation for the existence and continued development of open source software is hard to come by.

This paper presents a model for understanding behaviour of agents using and/or contributing to open source software. The model illustrates behaviour of agents under three licenses regimes: 1) The GPL, The BSD and 3) The Microsoft EULA. The latter license is not an open source license and is included as both a reference and as an example of a general situation where users do not contribute source code.

The model is uses economic theory of externalities and opportunity cost as a measure of agents' costs. The basic premise is that agents will only participate in development if there is a net benefit.

It is observed by using the model that the three licenses induce different incentives for both the agent producing and the agent consuming the program. Agents who develop and distribute software should carefully consider the

Note: This paper is based on my upcoming (currently under review) Ph.D. thesis. Short sections of text, no more than two paragraphs may be quoted without permission provided that full credit is given to the source. All rights reserved © Kasper Edwards November 2003.

Keywords: Software licenses, incentives, model, economics

---

[*] Kasper Edwards is a Assistant Professor who has recently finished writing a Ph.D. on open source software development and is currently working on product configuration systems. I would like to thank Ass. Prof. Jørgen Lindgaard Pedersen for helpful comments and suggestions on earlier drafts and I, of course, assume full responsibility for any remaining vulnerabilities.

1 Introduction

The existence and continued development of open source software (OSS) isindeed intriguing. Programmers situated all over the planet contribute to developing software, which sometimes reach a level of quality where it becomes the dominating software package in its category. The Apache web server software is one such program, which is dominating the market for web servers with little over 65% off the market [Netcraft, October 2003]. The Linux operating system is yet another example of a highly successful effort to produce some very capable software. If one takes a look at SourceForge or Savannah it becomes apparent that open source software is a phenomenon of considerable size and scope. In recent years the discussion of open source software has begun to emerge as a policy debate of whether the public sector should endorse it's use.

Open source software is defined by the restrictions or perhaps lack hereof in the licenses. In this paper OSS is associated with software distributed, as source code, under at license allowing: 1) Free[1] copying, 2) Modifications to the program, and 3) Free redistribution. A program distributed under such conditions allows a user to obtain the program, use the program as he may see fit, make modifications (here the presumption regarding source code is essential), make copies and distribute these.

At first sight from an economic perspective it would appear that there are little incentives for developing open source software. When a program has been developed at a given cost to the developer, users are free to copy, modify, and distribute the program. How is the developer going to make a living from such a license? Indeed the real question appears to be: What are the incentives to develop or contribute to the development of open source software?

This paper attempts to offer an explanation rooted in economics to this question. The explanation is derived from a model of software development and consumption, which tries to model the relationship between developer and user. The model is based on the hypothesis that behaviour of agents can be deduced from properties of software and properties of the agents involved in the exchange.

In the quest of doing so section 2 present three licenses are chosen as examples of software with distinct licenses with different properties, which influences behaviour. Section 3 analyse the individual licenses, briefly present two types of agents which are part of the model and the roles the agents can assume are presented in section 4. Section 5 then proceeds to develop a model of software development and consumption. The conclusion is presented in section 6, which sum up the paper and introduce an important discussion about the implications of this understanding for policy.

2 Three Licenses

Before developing a model license properties must be understood i.e. what behaviour is allowed by a license. A vast number of open source software licenses and a quick glance at

---

[1] Free in the sense of freedom not cost.

opensource.org reveal that 48 licenses are currently approved as open source licenses. An approved license is a license, which conforms to the open source definition[2]. While not bothering to go into the details of the open source definition the following description is sufficient for our purpose: A program distributed under an open source software license allows a user to obtain the program, use the program as he may see fit, make modifications (here the presumption regarding source code is essential), make copies and distribute these.

From these 48 licenses three licenses have been chosen as the basis for the model. The three licenses are: 1) the General Public License (GPL), 2) the Berkeley Software Distribution (BSD) license, and 3) the Microsoft End User License Agreement (MS EULA). The GPL and the MS EULA have been chosen as they represent two opposites of a spectrum going from free to proprietary software. The BSD license represents a middle point where it is possible to turn the program into a proprietary program. The Microsoft EULA is a typical closed source software license offering only the right to use the software, and a very limited and perhaps inconsequential warranty. There is no right to copy, distribute, or modify the software, and only a limited right to sell the software to third party.

The GNU GPL on the other hand requires the source code to be available, and allows people to use, copy, distribute, and modify the source code. It is, however, required that all changes, which are distributed, are made publicly available. The GNU GPL also includes the so-called viral effect, which requires that a program, incorporating code from a GPL'ed program, becomes GPL as well.

The choice of the third license is less straightforward. It must, however, be an open source software license, and it must be less strict than the GNU GPL. The BSD license is proposed as the third choice, and unlike the GPL, the BSD license

- Does not require source code to be available, when making a binary distribution
- Does not require modifications to carry the original license
- Requires that derived works must carry a different name
- Can be mixed with code carrying a different license
- Does not restrict costs of supplying source code.

Although both licenses are open source software licenses following the Open Source Definition, they are different beasts' altogether. The BSD license allows far greater degrees of freedom in the sense that individuals or firms can use the licensed code in closed source software. In contrast, the GPL license provides far more freedom in terms of ensuring that the program remains free and open source. The BSD has no requirements about modifications having to be distributed and the BSD-license places no restrictions on using the code in other projects.

---

[2] http://opensource.org/docs/definition.php

3 Two Types of Agents

When observing open source software development it is evident that some developers are individuals who participate in their spare time while other are professional programmers paid for what ever reason to contribute to a given project. For this reason a model of software development and consumption must be able to capture and analyse both types of agents.

Firms and individuals are expected to have different incentives, as firms pay salaries and must consequently generate profits. Individuals do not rely on open source software development as a source of income.

The key theoretical difference between the two is that firms are profit maximising, and individuals are utility maximising. When individuals' utility maximise, they seek to strike a balance between work and leisure (in this case contributing to OSS projects). When working too much, there is no time for leisure, but when working too little, there is no money for leisure. The difference between the two agents is most obvious, when the two types are placed in a situation, where choices have to be made. The firm is assumed to always make a choice based on profit expectations, whereas the individual will also value other benefits that might be gained, such as the joy of working with something interesting.

3.1.1 Resources and Incentives

Firms and individuals differ in available resources, in incentives for using and/or developing, and in the goals for using and/or developing open source software.

Firms are able to commit the amount of resources necessary to complete the desired task. Needless to say, firms have to act within the boundaries of economic capability and expected return on investment. Open source software may be free of charge, but implementing a solution to a problem certainly requires an investment in terms of time and training of users.

In contrast, an individual can only commit his personal time. If unemployed, the individual can spend all his time developing open source software, but only his time. Individuals cannot allocate other individuals' time to complete a desired task.

It is assumed that individuals do not have a direct profit incentive, and that they rely on other activities i.e. a job to generate income. Thus the resource available to an individual can be no more than his personal spare time. For this reason we use opportunity costs to measure the cost of an agent developing a program or helping to develop the program. Opportunity costs are defined at the cost of a resource, measured by the value of the next best, alternative use of that resource [Stiglitz 1996:a16]. For an employed person developing a program in his spare time the opportunity cost is equal to the cost of working in his spare time. Assuming individuals whish to spend their spare time not working the cost must be higher than regular salary. Should the individual be hired by an employer to develop open source software, this very individual would then be considered a part of a firm and therefore adhere to the logic of a firm.

Incentives for using and developing open source software differ between firms and individuals. Lerner & Tirole [2000] pointed out that signalling incentives could be part of the incentive for individuals developing open source software, and that this would serve them to acquire a future job. Firms might also have signalling incentives, however, these are small compared to the individual. Eric S. Raymond [2000] pointed to reputation effects as the incentives for participating in open source software development. Reputation effects were of value to individuals, who sought to gather a pile of reputation, which could be used to attract other developers to the individuals' own projects. Firms were not mentioned to be part of the reputation game. Firms might have a cost minimizing incentive to develop open source software. If a program developed by a firm, perhaps for internal use, were interesting to developers who were not employed by the firm, then the firm would gain a productive resource for free. One could also imagine a political incentive to use open source software i.e. discontent with a monopolist software vendor. Firms are not able to afford such leisure, unless political consumers pressure the firm, in which case there is a market incentive. Both firms and individuals have a cost saving incentive to use open source software. Open source can be copied freely, and there is no licensing cost for using it. Access to source code and the possibility of being able to modify programs to suite particular needs, are also incentives, which relate to both firms and individuals.

The next section presents two roles that are part of open source software development.

4 Two Roles and their Desired Use-product

Without further ado this section presents some terms which are central to the model. The entity, be it a firm or individual, who is responsible for distributing a program or some software is referred to as the maintainer. Firms and/or individuals who use a program or software is referred to as users. However, for open source software it is possible to contribute to the development and the firms or individuals who contribute to development is referred to as user-developers.

A user-developer is an agent who is a user and potentially a developer [Johnson 2001]. In this view a user-developer always begins as a user and for reasons, which will be explored here, may become a developer who contributes to developing a program. The user-developer term is tied to open source software development by Johnson [2001]. However, this restriction is relaxed here, and user-developers may exist for software licensed under each of the three licenses. The reason for relaxing the restriction is to allow for a consistent analysis of the three licenses without having to separate the analysis of the MS EULA into a different type of analysis.

A use-product is an agent's particular use of a combination of features in a program. Software is not a simple good like grain. Software is a complex good in the sense that a software good consists of many different features working together [Bessen 2001:1]. Agents have different preferences and will desire different features, and in turn agents will use different combinations of features in a program. By containing many different features in one single program, the supplier tries to satisfy the aggregated demand while actually serving

many discrete markets. A program containing a number of different features represent a number of use-products equal to the number of possible combinations of features. Given the individual preferences, each agent will demand only a single use-product i.e. only a single combination of features. It is important to understand that the notion of a use-product does not constitute a neither/nor disposition, where a program is rejected because the exact number of features desired is not present. A user may desire a use-product consisting of n features of which only n-1 features are present in the program. Given no alternative program containing the desired use-product, the user will have to make do with a use-product consisting of n-1 features. While an agent desires the complete use-product, the agent will maximize his benefits by choosing a good containing the most important features. A user will not choose not to use a program at all and the features it may offer, just because one or more of a large number of features is missing.

The term "user" used in the previous chapter were sufficient for illustrating the simple situation analysed. However, in this chapter a model is developed, which allows users to make and distribute modifications to some of their software. For this reason we shall use the term user-developers, which describe an agent who may choose to use or make modifications to a program (assuming the license permits).

The maintainer is still the agent who is distributing the program. Within the open source software community the agent, who is responsible for releasing new versions of a program, is referred to as the maintainer, and we shall continue this tradition. Lerner & Tirole [2000] has referred to this person as the leader of a software project, but in the eyes of this author a maintainer is unable to lead in a traditional sense. The maintainer in an open source software project has no means available to direct the efforts of user-developers. See Edwards [2001] for an in depth discussion of this issue.

In the next section a model is developed, which draws from the previous sections.

5 A Model of Software Development and Consumption

In this section a model of software development and consumption is developed by deducing the behaviour possible for the maintainer and user-developers. Behaviour is deduced by analysing the three licenses, which begins by a brief presentation of the most important characteristics of the license. Based on the characteristics of the license the model of software production and consumption is expanded with the relevant feedback loops and other agents who might participate in production and consumption. This is done in a sequence of steps, where the appropriate elements are identified and then added graphically to the model. Having developed the model, the incentives for the maintainer and user-developer and their relationship are analysed. Lastly, the dynamics generated by imagining the model being an image of a continuing process is analysed.

5.1 The MS EULA license

The MS EULA does not permit reverse engineering, and software distributed under this kind of license does not come with source code. This type of software is only distributed as

binary programs ready to run on a computer. The license only allows use of the software, and use is restricted to installation on a single computer, the implication being that only one person can use the software at the same time. The license explicitly prohibits copying and distribution of the program covered by the license.

The relationship between maintainer and user-developer mandated by the license are one where a maintainer develop and distribute a program. If a significant part of the desired use-product is contained in the program user-developers will have an incentive to obtain (buy) the program in which case the user-developer engage on what is defined as *private use*. Private use is simply that the user-developer is confined to just use the program and the use-products it may offer. It is not possible to extend the number of use-products contained by adding features.

If the desired use-product is not completely contained in the program, the user has an incentive to further pursue the desired use-product from the program. We can imagine two general types of problems with obtaining the desired use-product from the program: 1) Missing features and 2) Faulty features.

A missing feature is a feature not present in the program, and no combination of features or other manipulation of the program will produce the desired use-product.

A faulty feature, on the other hand, means that a feature is present but not functioning correctly, and the desired use-product cannot readily be obtained. In order to obtain the desired use-product the user must either research the program himself or consult other Agents who use through user-to-user assistance e.g. a user forum.

The relationship between the maintainer and users can be depicted as shown in Figure 1:

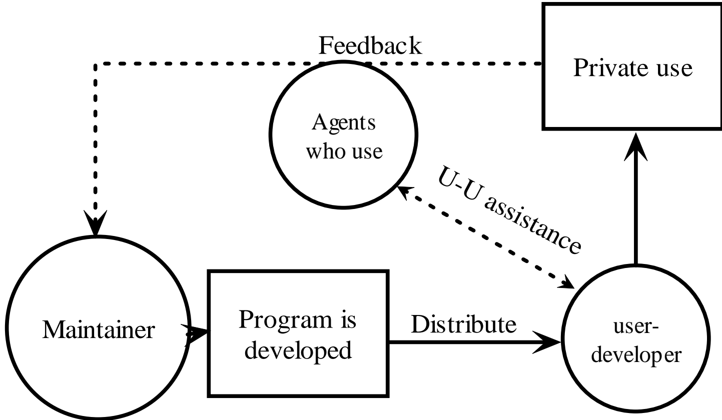

Figure 1: Programs distributed under a MS EULA style license restrict modifications and only allow a simple feedback mechanism to the maintainer of the program and other users who may engage in user-to-user assistance.

In light of the above figure we now turn to discuss the economics involved then the actions available to the maintainer and the users.

5.1.1 Analysing the Maintainer

The maintainer has a choice between developing a program or not, and if the program is developed, the maintainer must decide whether to distribute the program. For simplicity we

assume that there are no costs associated with distributing the program. Under this condition the choice of whether to distribute the program or not is a non-issue, as the maintainer will distribute, if the program is developed.

The situation we are analysing is a simple market situation, where a maintainer must decide whether or not to offer a product on the market. The maintainer will offer the product for sale, if he estimates that the cost of production including opportunity cost can be covered.

We know that software is a special good with no physical existence, which only has an existence on a computer, and the cost of production i.e. the act of reproducing the program is negligent compared to the cost of development. When developed, a program can be reproduced in countless copies at insignificant cost. The cost of developing a program is then equal to the opportunity cost associated with the time spent on development. The maintainer will choose to develop the program, if he is convinced that future sale of the program may cover his opportunity costs.

The maintainer will benefit from feedback externalities, if users decide to provide the feedback. The feedback represents information about missing or not functioning features, which users desire. If the maintainer implements the missing feature or repairs the defect feature, there is a possibility of enjoying an increase in sales. However, the maintainer will also actively develop the program to increase sales. The maintainer is offering the program for sale in competition with other programs, which induce an incentive to improve the program to satisfy both existing and new users (customers).

The maintainer will suffer a cost, if he decides to implement the desired features. The maintainer will implement the features, if the cost is expected to be less than the additional income generated by offering the features.

Feedback from users is welcome, as it reduces the cost of developing new features and also ensures that the features implemented in the program are actually wanted by users. While there is a feedback externality present in the relationship between maintainer and user, it is not the driving factor for distributing new versions and motivating the maintainer. The maintainer has chosen to create a club good and profit from selling this club good, and it is the profit from selling the program, which is the driving factor. The feedback externality is merely an added bonus, which the maintainer may decide to use.

5.1.2 Analysing the User

The user interacts with both the maintainer and perhaps also other agents who use the program. The two types of interaction are different and must be treated separately, and we shall begin by analysing the relationship between the user and the maintainer.

The user has a choice between buying the program distributed by the maintainer or not. If the program is bought, the user can then further choose between engaging in one of the two feedback loops or not. This being a standard market situation, the user will buy the program if the expected utility is equal to or higher than the price.

5.1.3 Dynamics

In general the relationship between maintainer and user is that of a market with a feedback loop. Users buy programs developed by the maintainer at a price, and users may offer feedback to the maintainer thus creating an externality. The maintainer is driven by a profit incentive and will use the feedback to improve the product and thereby increase sales. In this relationship the dynamics, which drive development and consumption, is driven by an invisible hand. The feedback provided by users is technically an externality, as the users are not compensated nor encouraged to do so. While the externality is welcome, it is none the less a secondary effect. If the maintainer does not improve the product, another maintainer would satisfy user's demand by providing a competing program.

User-to-user assistance is on the other hand driven by externalities. Each user answering a question is generating an externality, the value of which is further sustained by the searchability of Internet based forums. Users offer answers to question in the anticipation that they may need help later, and offering help is a way of signalling that it is worthwhile to help this user. It can also be argued, as done by Lakhani and Von Hippel [2000:35], that users providing answers to questions perceive that value of their answers as close to zero and thus not worth protecting. However, Lerner and Tirole [2000] argue that open source software developers are motivated by signalling incentives, which may also help explain why users provide help to other users. By doing so a user signals a strong competence in solving a certain type of problems. Offering help also signals competence, which is the desired good in these forums, and hence users gain social status. This assumes that some degree of stability exists in the forums, as newcomers will be unaware of who has knowledge and who does not. However, the status derived when providing help is likely to be contained in the narrow forum defined by the community of users of the program.

The conclusion is that for the MS EULA feedback externalities have little impact on the dynamics. The behaviour of the maintainer and the user is by and large directed by simple market mechanisms.

5.2 The GPL License

The GPL license is far more permissive than is the MS EULA, and this has major implications for the model of software production and consumption. Unlike the MS EULA, the GPL allows the licensee to copy, modify and distribute a program licensed under the GPL. It goes without saying that the GPL license allows for unrestricted use, and the user may use the program on as many computers as the licensee may please. The GPL license requires the source code to be available, and thus users are able to make modifications to the source code, and for this reason users become user-developers.

Looking at this description it immediately becomes apparent that programs licensed under the GPL must resemble a pure public good. The license allows for copying and distribution, which essentially ensures that programs are non-excludable.

When a maintainer has released[3] a program under the GPL license, the program is available to anyone interested. Any user-developer may now obtain the program and use, copy, modify, and distribute the program. When engaging in private use, the GPL license offers the same feedback mechanisms as the MS EULA, and user-developers may contact the maintainer and/or other user-developers.

The GPL license further allows user-developers to modify, copy, and distribute the program. In this analysis we are only concerned with the right to modify and distribute the modified version. The right to copy is a necessary prerequisite for modifying and distributing a program, and the analysis will not treat copying separately. A user-developer may very well choose to distribute identical copies of the program, which he received and is thereby re-distributing the program.

We are not concerned with the possible choice of re-distributing an unmodified program, as the program is already assumed to be available to user-developers as a download from the Internet. We are, however, interested in user-developers, who spend resources modifying a program. If a user-developer makes modifications to a program, it triggers a clause in the GPL license, which makes a sharp distinction between whether the user-developer keeps the modifications private or decides to distribute the modifications.

If the modifications are kept private, the GPL license does not require the source code for the modifications to be available for other agents. A user-developer can obtain a program distributed by the maintainer, modify the program, and keep the modified program private. These two choices: 1) Modify the program and 2) Keep the modifications private are added to the model of the MS EULA in Figure 1. The two choices must be seen as closely connected, as a user-developer upon decision on making modifications to a program must further decide whether to keep or distribute the modifications.

While a user-developer may distribute the program to whomever, he may desire, it is assumed that modifications distributed are returned to the maintainer. This is an important assumption, as it adds a feedback loop from the user-developer to the maintainer. Unlike the feedback provided from private use to the maintainer and/or other agents who use, this feedback loop contains modifications to the program. Such modifications may be added features, a new use-product, which a user-developer has desired, or a modification, which fixes a defect use-product.

Like user-developers, who just use the program and engage in user-to-user assistance to solve problems with their desired use-products, a user-developer who modifies the program may also engage in similar activities. For ease of understanding, a user-developer who has modified a program shall now be referred to as a developer, and user-developers who only use shall be known as users. Developers like users have a desire to discuss the problems and ideas they encounter while modifying a program. However, as users provide non-source code feedback, and developers provide source code feedback, they are unlikely to find help for

[3] It is assumed that a release does in reality mean making the program available for download from the Internet.

their problems in the same forums. Developers and users need separate forums to discuss their problems - a forum where developers may discuss ideas and problems related to making modifications. This forum has the special property of the maintainer participating. The maintainer is participating in this forum because the other developers are making modifications to the program originally released by the maintainer. The maintainer being responsible for releasing new versions of the program will be interested in discussing modifications to “his” program. If the maintainer is not interested in discussing modifications and releasing new versions containing modifications from developers, the developers may very well release a new program containing the modifications.

This results in Figure 2 presenting the model of software development and consumption under the conditions for the GPL-license.

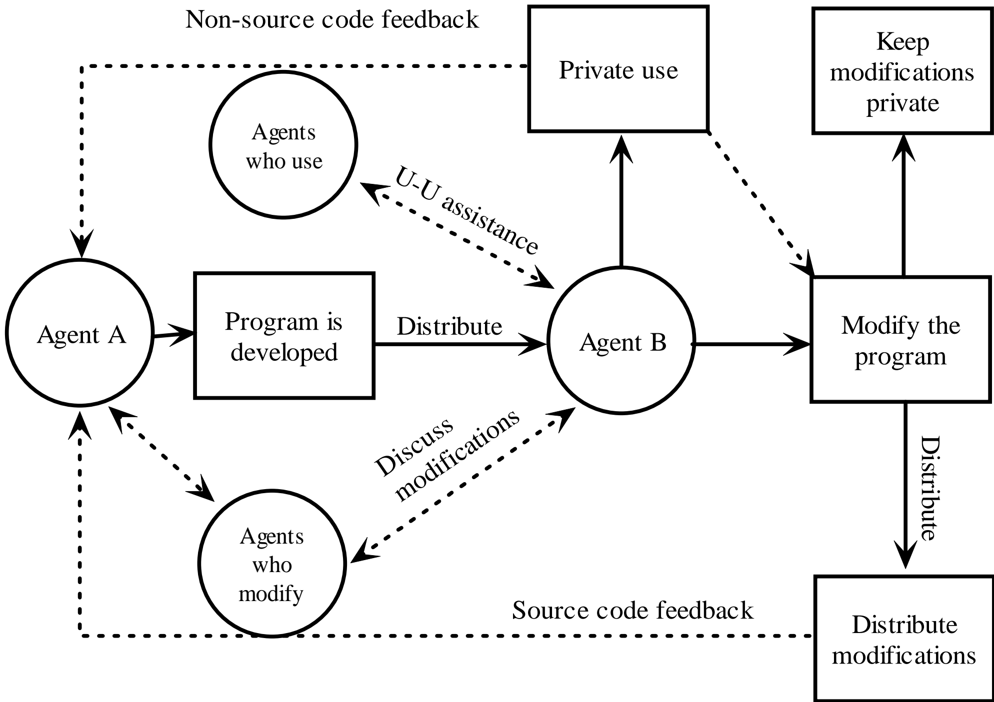

Figure 2: When other user-developers are making modifications to the program user-developers will discuss their modifications in the forum for Agents who modify.

5.2.1 Dynamics Induced by the Cost of Being too Late

The analysis of the maintainer revealed that there are incentives for the maintainer to develop and distribute a program under the GPL. There are also incentives for the user-developer to make modifications to the program and redistribute these. However, the incentives are mostly derived from the maintainers and user-developer’s private need, and the focus has so far neglected to take dynamic effects into account. A brief insight into the dynamics was seen in the last situation, where the user-developer distributed his modifications in order to avoid having to implement his modification each time a new version of the program was distributed by the maintainer

Dynamic effects arise, when the process of software development and consumption is repeated many times and becomes a continuing process, where the maintainer and user-

developers continuously distribute their modifications. This principle can be illustrated using the concept of goods, where the size of the available good (Z) is equal to the sum of features (z) i.e. $Z = F(z^0, z^1, ..., z^n)$ where n is the total number of features in the program.

We now analyse a situation, where a maintainer releases a program containing a single feature z^0. The situation also involves two user-developers, who obtain the program. Both user-developers desire a use-product not contained in the program, and this provides an incentive for making modifications to the program. For the sake of clarity, the two user-developers will be named UD1 and UD2. UD1 desires a use-product requiring the feature z^1, and UD2 desires a use-product requiring the feature z^2. In order to obtain the desired use-product both UD1 and UD2 must make modifications to the original program. However, both desired use-product require modifications made, which are in the same area of the source code. This is an important premise, as the two modifications mutually exclude the other from being integrated into the original program without significant additional costs from integrating the two modifications.

UD1 is the agent who is the first to distribute his modifications to the maintainer. The maintainer finds the modifications interesting and good enough for inclusion in the program, and subsequently a new version of the program containing the modifications are distributed. The size of the available good in this new and modified version of the program is $Z = F(z^0 + z^1)$.

At the same time the maintainer distributes a new version containing z^1, UD2 has finished creating z^2 and distributes his modification to the maintainer. The maintainer receives the modification from UD2 and observes that the modifications are incompatible with the new version of the program, and implementing the modification from UD2 will break the modifications made by UD1.

This leaves the maintainer with a choice between 1) Reject the modification from UD2 or 2) Rework the modification and integrate it into the program. The difference between the two choices is a matter of who will have to cover the cost of integrating the two modifications. From the maintainer's perspective, the first choice has no cost other than the cost of answering "No" to UD2. The second choice results in the maintainer becoming responsible for integrating the modifications, and he must then cover the opportunity costs associated with the time spent on making the integration. The maintainer has an obvious incentive to reject the modifications from UD2 in order to minimize his own costs from maintaining the program.

UD2 is now faced with a potentially sunk cost, as the modification will not be integrated in the program by the maintainer. If UD2 is still interested in using the particular use-product, which he has developed, he must continue to use his personally modified version of the program, which in the model is equivalent to keep modifications private. However, doing so

has consequences and UD2 cannot enjoy the benefits of new versions of the program, as they are incompatible with the particular use-product that he desires. This leaves UD2 with a program, where the size of the good is $Z = F(z^0 + z^2)$. Note that the feature z^1 is not present.

UD2 may choose to make modifications to the new versions of the program and integrate his personal use-product. This would create a program, where the size of the good is $Z = F(z^0 + z^1 + z^2)$. Naturally, UD2 suffers a cost from doing so, and each time a new version of the program is released, UD2 must decide if the new version of the program offers advantages, which justify implementing his personal use-product in the new version of the program.

UD1 on the other hand is in a position, where he gets the best of both worlds. His modifications are integrated in future versions of the program, and he can anticipate the benefits of new versions of the program. As new versions of the program contain UD1's particular use-product, he does not have to make modifications to the program.

A rejection signals not only to UD2 but also to other user-developers that they will have to distribute modifications, which are based on the latest version of the program. In this perspective, a maintainer's rejection of a modification becomes the source of dynamics in the model. Rejection of modifications imposes costs on the user-developer who has implemented the modifications.

The unfortunate situation between UD1 and UD2 could perhaps have been resolved though coordination. Given that both user-developers were seeking to obtain similar but not identical use-products, it is likely that a solution satisfying both could have been created. UD1 and UD2 should then have signalled their intentions to make the modifications in the forum for agents, who make modifications. While UD1 and UD2 may have signalled their intentions, this does not ensure coordination. Signalling is costless and making the actual implementation is not, and since agents have no way of guaranteeing their commitment, such signals may not be taken seriously. UD1 and UD2 cannot be sure when either of them will deliver, and therefore there is a high degree of uncertainty regarding the two agents' intentions.

If the two agents were to present a preliminary version of their modifications in the forum for agents who modify, they would both be able to signal their intentions by presenting their modifications. This would also allow the two user-developers to offer comments and suggestions for changes, before the cost of changing the modification becomes too high. Over time UD1 and UD2 will learn from observing each other's behaviour and whether to trust promises about future modifications.

A program rarely has only two user-developers, and the more user-developers the greater the possibility that someone is making modifications to the program. The more use-products present in the program, the more appealing will the program be. This has the additional effect of increasing the likelihood of user-developers making modifications to the program. With

each new version of the program more use-products are contained in the program, and the more appealing it will be. When a user-developer obtains the program and discovers that 99% of his use-product is contained, he may be willing to modify the program, adding the last 1% so that 100% of his use-product is contained. Whether he makes the modification or not will depend on the cost of making the modification compared to the cost of engaging in strategic waiting for someone else to make the modification.

A user-developer, who engages in strategic waiting, cannot know when another user-developer will implement the last 1% of his desired use-product and thus the user-developer cannot predict the cost of waiting.

This process of improving the program and releasing new versions gradually lowers the barriers for modifying the program. The lower the cost of obtaining the desired use-product, the lower the barrier to making modifications. However, here we begin to touch on the technical issues of programming and making modifications to a program, which are not readily contained in the model of software production and consumption. The essence of the argument is that the source code for a program can be structured in a variety of ways, of which some are easier to comprehend than others. As program grows, it suffers the danger of becoming a large kludge, which only the maintainer can understand. If this happens, the barriers to modifying the program raise significantly, as user-developers will have to invest significant amounts of time in just learning how the program functions. This time is essentially unproductive and little fun, as the user-developer during this period is unproductive and is not modifying the program to obtain his desired use-product.

5.2.2 Low Profit Potential Inspire the GPL license

If a maintainer could a priori know that his program would eventually dominate a particular market segment, he would choose a different license allowing the option to create a club good. And here we touch on a central issue in the production and consumption of open source software.

One reason for distributing a program under the GPL license is because the profit potential, when releasing the first version of the program is close to zero. The first version of a program is usually very rudimentary and only containing the very core feature, which the maintainer desires. Such program resembles what Brooks' refer to as a little program, which has difficulties functioning outside the confinements of the development environment where it was conceived. Such a little program as a profit potential close to zero simply because of the problems users will experience trying to obtain the desired use-product.

The program has been developed to provide the maintainer with a particular use-product. At this point in time the cost of the program is equal to the opportunity cost of the time spent. It is assumed that these costs are covered if the maintainer can obtain his desired use-product. Depending on the characteristics of the maintainer the cost may vary. The opportunity cost of a student is quite small compared to an experienced professional. Regardless, the maintainer will only continue to develop the initial version of the program if the cost of doing so is equal or less than the expected outcome (his particular use-product).

It is beneficial to use the classification of programs by Brooks [1995] to understand the costs and benefits faced by a maintainer. In opposite ends of the scale of total cost of developing a program we find the little program and the programming systems product. The little program is a small idiosyncratic program, and the programming systems product is the complete program, which is well documented and tested. The two programs contain the same number of features and use-products for the maintainer, but the cost of the programming systems product is nine times that of a little program.

When a maintainer creates a program to solve personal need, it will be a little program given the cost of expanding the program into a programming systems product. The maintainer will have to invest nine times the effort in order to transform his little program into a programming systems product, which he may sell and profit from. A maintainer then faces a situation prior to distributing a program, where he must ask himself if he is willing to make a further investment in the program before distributing the program. If he is willing to make the investment and believes the program has sufficient sales potential, also considering the competition, he must wait and distribute under the MS EULA.

However, as the program is developed to solve a particular problem and provide a particular use-product for the maintainer, the profit foregone by distributing the program is close to zero when distributed as a little program. As a little program, it has no profit potential and must turned into a programming systems product in order to be sold.

The maintainer can on the other hand expect that if he distributes the program and it is adopted by user-developers who make modifications, he will receive a benefit. User-developers will enhance their personal use-product, and if it is overlapping the personal use-product of the maintainer, he will receive a benefit, the value of which is equal to the opportunity cost of the maintainer having implemented the use-product himself.

5.3 The BSD License

The BSD license is the last and the most permissive of the three licenses allowing even greater degrees of freedom than does the GPL. The BSD license distinguishes itself from the GPL by allowing the user-developer to distribute his modifications and a modified program under a different license altogether.

In terms of the model of software development and consumption, the BSD is similar to the GPL license with the addition of one choice for user-developers, who have decided to modify the program. Besides keeping the modifications private and distributing the modifications, the BSD license introduces a third choice: Distributing modifications as closed source. This additional choice of being able to distribute the modified program under a different license has implications for our understanding of the maintainer and user-developers. The additional choice can be seen in Figure 3

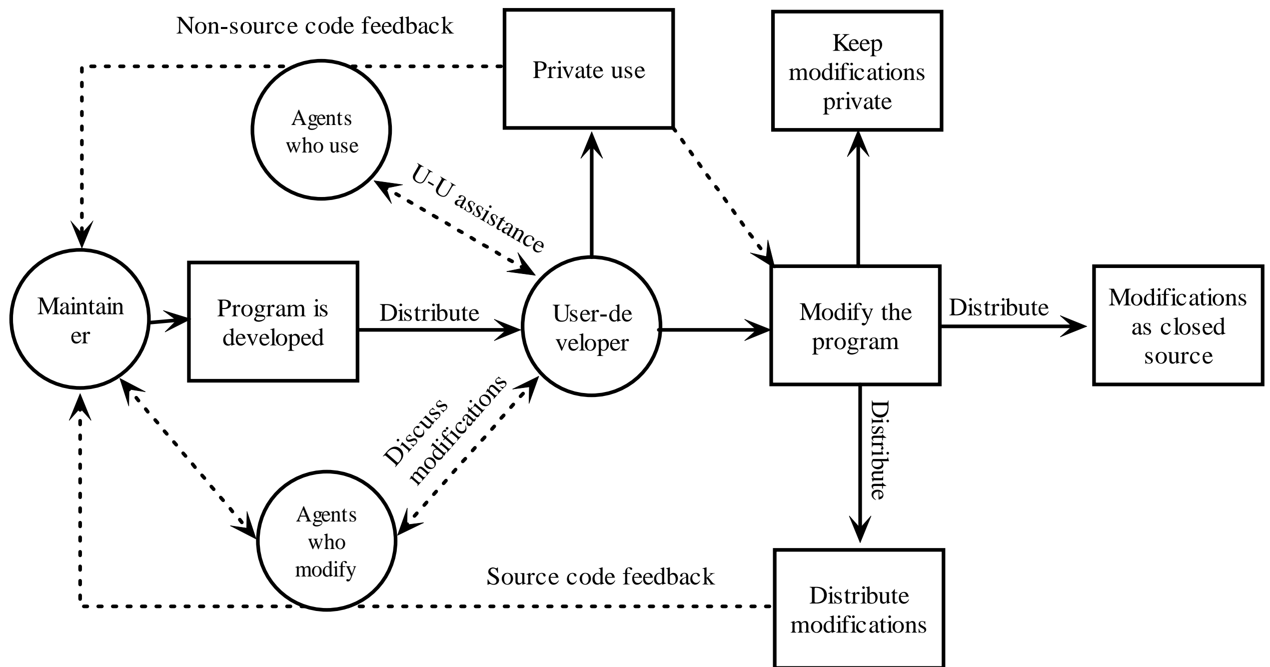

Figure 3: The BSD license introduces a third choice for user-developers, who make modifications to the progam: Distribute modificaions as closed source.

Given the complete model for software development and consumption under the BSD license we now turn to discuss the incentives for user-developers distributing modifications as closed source. The conditions for providing non-source code feedback is almost identical to the situation under the MS EULA, and the dissimilarities will be discussed first.

5.3.1 Creating a new MS-EULA program

The user-developers receive an additional choice when obtaining a program distributed under the BSD license. User-developers may choose to distribute the program under a completely different license such as the closed source MS EULA license.

When doing so the user-developer changing the license also changes role and becomes the maintainer of the program, which he just distributed. The user-developers of this program in turn become user-developers of his program. The relationship between the maintainer of the original program, the maintainer of the distributed

program, and its user-developers can be illustrated as in the following

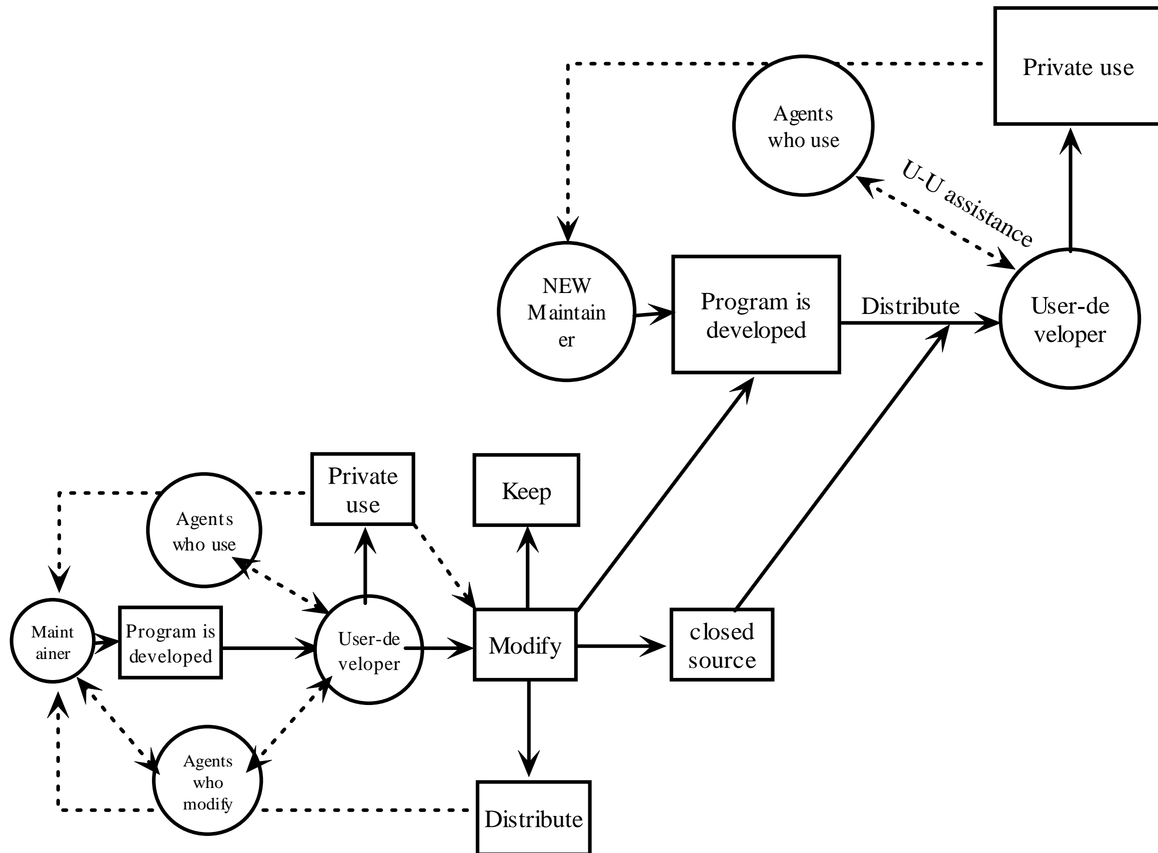

Figure 4: When a user-developer modifies a program and distributes the progam as closed source, he creates a separate instance of the model for software developerment and consumption. The two arrows between the two models illustrate how they are interconnected.

:

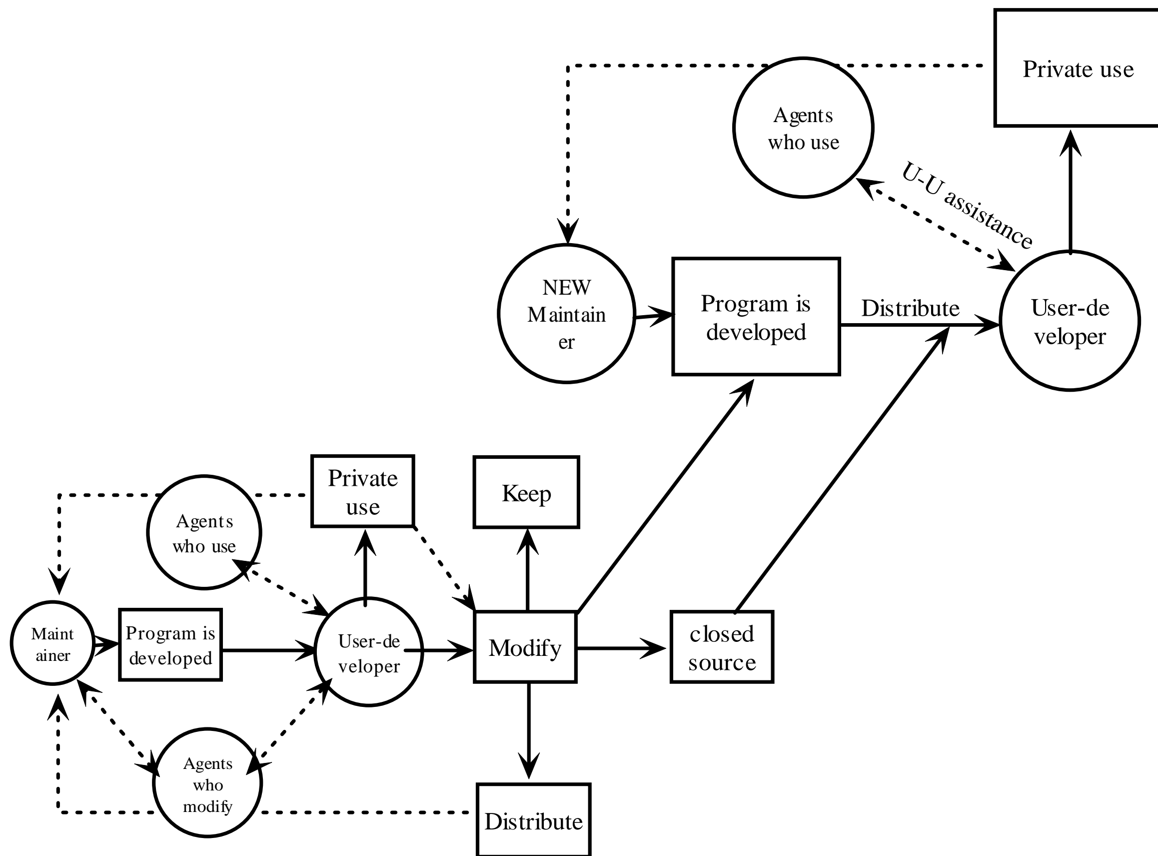

Figure 4: When a user-developer modifies a program and distributes the progam as closed source, he creates a separate instance of the model for software developerment and consumption. The two arrows between the two models illustrate how they are interconnected.

When the user-developer modifies the program to be distributed as closed source, he is entering the MS EULA model and becomes the “NEW maintainer”. Modifying the program compares to the “Program is developed” situation in the MS EULA model, and “Distribute as closed source” compares to “Distribute” in the MS EULA model.

After having distributed the new version, the relationship between user-developers of the new program is disconnected, as the source code is not available and the license prohibits modifications. The NEW maintainer may, however, continue to import modifications from the original program into his program.

The program distributed by the original maintainer begins as a public good, which is an externality. The original maintainer is in no way compensated, when the user-developer distributes the program as closed source and thus transforming the program into a club good. Indeed it is fascinating that the BSD license allows the one public good to be transformed into a club good without any compensation. The NEW maintainer may even continue to use modifications from new versions of the original program to improve his own program. It deserves to be mentioned that the GPL does not allow such behaviour. The viral effect of the GPL ensures that a modification from a GPL program inserted into a differently licensed program changes the license of the combined program to the GPL.

5.3.2 Dynamics – The BSD License

The BSD license has implications on the dynamics in the model. The dynamics of the GPL were tied to expectations formed between maintainer and user-developers. These expectations were tied to the fact that the GPL did not allow distribution of closed source modifications. The maintainer and user-developers of a GPL program could rest assured that if someone made modifications worth distributing, the modifications had to be publicly available. Indeed the GPL license allowed user-developers to keep their modifications private, however the cost of maintaining them ensured that only few would do so.

The BSD license on the other hand allows user-developers to obtain a program, which is a public good and turn it into a club good without incurring any cost. This means that any user-developer may build a business on distributing closed source programs.

This changes the expectations between user-developers and the maintainer as a whole. User-developers may suspect that the maintainer or other user-developers are just waiting to capitalize from their work. It is easy to imagine the reaction from the maintainer and user-developers, who have distributed modifications, if a user-developer form a successful company based on the program. They are very likely to feel exploited.

Thus it must be expected that the BSD license, compared to the GPL, has lower incentives for user-developers to make and distribute modifications to the program. It is actually possible to imagine a situation, were user-developers make and distribute modifications to the program in anticipation of distributing the program as closed source when sufficiently matured. This would be a situation, where user-developers compete to be the first to market a closed source version of the program.

If a user-developer modifies and distributes a closed source version and becomes maintainer, he still has incentives to make contributions to the original program. It is known that maintaining a closed source program requires a significant amount of resources, and the maintainer will not receive modifications from user-developers. The maintainer may, however, still include modifications from the original program into his closed source version.

The maintainer of the new version still has an incentive to distribute some of his modifications back to the original program. For understanding why, it serves our purpose to distinguish between two types of modifications: 1) Modifications which adds a competitive advantage to the program and 2) General improvements.

Modifications, which add a competitive advantage to the new program could be an improved installation process that would let customers obtain their desired use-product without problems. Another example would be an improved user-interface, which would let customers obtain their desired use-product easily. In general it is only possible to identify modifications, which add a competitive advantage, by analysing the original program with the intended customer segment.

General improvements, on the other hand, do not add new features but will often entail improvements to existing features and fixes to defect features.

The maintainer will generally distribute modifications, which improve the infrastructure on which the new and added features rely. Having the modifications integrated in the original program lowers the cost of having to maintain these features personally.

This highlights the difference between the GPL and the BSD license. The GPL does not allow this kind of commercial exploitation of the original program. On the other hand the GPL thereby excludes itself from being used in places, where there are incentives to keep parts of the modifications private and distribute others.

The maintainer of a program faces a trade off when choosing the license, under which he wishes to distribute his program. The GPL ensures that modifications, which are worth distributing, have their source code publicly available. The BSD ensures that anybody may use the program for whatever they want, and they may (but do not have to) return their modifications to the maintainer.

6 Conclusion

This paper has developed and presented a model of software development and consumption. The model illustrates the relationship between a maintainer and a number of user-developers within a single software project. A software project can be interpreted as an open source software project or a commercial software package. The model uses three licenses, the GPL, the BSD and the MS EULA as samples to illustrate how behaviour of agents is influenced by the properties of the respective licenses.

The important conclusion to be drawn is that maintainers are in a unique position, prior to distributing their program, to choose which kind of behaviour they wish to induce. In this respect maintainers may use the license as a means of stimulating incentives. The three licenses presented here only show three possible license regimes and the behaviour they induce. Maintainers are free to create other licences and indeed maintainers do so as is evident in the 48 approved open source licenses. However, there is an obvious danger of use-developers not understanding the subtleties of each of the individual licenses they come across.

While this research has focused at the level of the individual software project, resembling a micro level, there are significant implications at the macro level. Governments are key players when it comes to purchasing software and policymakers should leverage this power to demand licenses which besides the mere software purchase benefit the economy they are in office to protect and stimulate. Clearly, MS EULA type licenses have a built-in monopoly creating effect, which are unacceptable – unless you are a net software exporting economy.

A monopoly capable of controlling the standards on the basis of which the programs exchange files can effectively control a market and thus prevent new firms from entering the market. While it might be problematic in several respects to demand all software for the public sector to be some kind of open source license, demanding open standards and

interfaces will go a long way of allowing competition to grow. This will allow local software firms to enter the market and be a productive part of the economy.

There are substantial differences between the GPL and the BSD license and each has its strengths and weaknesses depending on the situation. The BSD allows for easy penetration as firms do not have to worry about open sourcing their intellectual property and this is also its weakness. A program may be of such a quality that firms eye a potential source of revenue in which case they add a few distinguishing features and sell the product. Problems arise as these closed source versions over time grow incompatible, which is damaging to the whole family of programs. Users have problems choosing the correct version and the third party developers refuse to develop add on software as long as no one has agreed on a standard.

7 References

Bessen, J., 2001, “Open Source Software: Free Provision of a Complex Public Good”, [Website] http://www.researchoninnovation.org/opensrc.pdf.

Brooks F. P., 1995, “The Mythical Man-Month - Essays On Software Engineering”, 20th Anniversary Edition, Addison-Wesley Longman.

Edwards, K., 2000 “When Beggars Become Choosers”, First Monday, volume 5, number 10 (October 2000), URL: http://firstmonday.org/issues/issue5_10/edwards/index.html

Lakhani, K, and Von Hippel, E., 2000, “How Open Source Software Works: “Free” user-to-user assistance”, MIT Sloan School of Management Working Paper #4117

Lerner, J. & Tirole, J.,2000, “The Simple Economics of Open Source”, Working Paper 7600, National Bureau of Economic research, 1050 Massachusetts Avenue, Cambridge, MA 02138

Johnson, J. P., 2001, “Economics of Open Source Software”, [Website] http://opensource.mit.edu/papers/johnsonopensource.pdf

Netcraft: http://news.netcraft.com/archives/2003/10/index.html

Raymond, E. S., 1999a, “The Cathedral and Bazaar”, in The Cathedral & the Bazaar, O’Reilly and Associates.

SourceForge, Website, http://sourceforge.net/

Savahanna, Website, http://savannah.gnu.org/

Stiglitz, J. E., 1996, “Economics”, Norton, W. W. & Company, Inc.