Open Source Software Development – When Free-Riding is not an Option

By Kasper Edwards[*]

Department of Manufacturing Engineering and Production, Technical University of Denmark
Building 423, 2800 Lyngby, Denmark. (email: ke@ipl.dtu.dk)
Document date: January 2001

ABSTRACT

Open source software can be viewed as a privately produced public good. Conventional theory holds this type of good to be subject to massive free-riding.

This paper explains the concept of free-riding and discusses free-riding in the context of open source software development. The paper argues that the current understanding of free-riding in open source software development is misleading.

It is proposed that people contribute open source software development and does not free-ride because the received good is not sufficient to allow free-riding. As a consequence features are added to the system , at which point the developer has the choice of keeping the changes or contributing back to the community. It is demonstrated that developers gain a long term advantage by contributing back to the community.

Kasper Edwards is a Ph.d. student at Department of Technology and Social Sciences at the Technical University of Denmark. The title of his project is “Technological Innovation in Software Industry” which is focusing on the Open Source Software development process. Kasper Edwards has a background as a M.Sc. (ENG) and has worked for several years as an independent computer consultant. He is an experienced user and administrator of both Linux and the different flavours of Win9x and NT/2000. As a hobby he maintains a PC gaming network in a youth club powered by Linux servers. As part of the project and hobby Kasper Edwards is engaged in the SSLUG (Skåne Sjælland Linux User Group, a 5000+ member user group) and participates in both technical and political discussions.

[*] I would like to thank Associate professor Jørgen Lindgaard Pedersen and dr. YYYY

1 Introduction

The understanding of the open source software phenomenon still seems to be in the early stages. Many different types of theory have been applied to different types empirical observations.

There are several aspects of free-riding and open source software development there has to be examined. How should we understand the concept of free-riding in the open source software world.

One question is whether users of OSS should be considered free-riders? Are a user a free-rider or are one free-riding when not contributing back to the community.

Is there a difference between free-riding in open source software and other types of material goods.

The conventional wisdom holds free-riding to bad since the some one are receiving a good without paying for the service. This make other people contribute less to producing the good and as such the good is undersupplied below social optimum. Is this the case for open source software. I OSS supply is not the problem rather new features – innovations are the problem. Is there a distinction between a good which has a marginal cost of production and a good like open source software that has zero marginal cost of production?

Is it possible to free-ride when the marginal cost of production is zero?

The marginal cost of innovation is NOT zero and people might free-ride for new innovations. In this situation people would wait for a new innovation to appear instead of implementing the innovation. Problem: People will not know how long they must wait before the wanted innovation appears.

There are indications that projects or interest groups are formed around questions and personal need for a given functionality. There are many examples of people arsking for a feature in a forum and other people answering that they have a home built implementation of the requested feature which, they can have for free. Often this spurs a back and forth of ideas, suggestions and usually implementations are also presented. In this case a question about features quickly turns into a specification of the needed features and a description of what and how to make the implementation.

2 Theory - Free riding in the books

(Crones and Sandler 1996, p.30)

1) It refers to the sub-optimality that characterise a Nash Eqlibrium
2) The failure of individuals to reveal their true preference for the public good through their contribution.
3) The tendency for public contributions to decline as group sizes increase.

for helpful comments and suggestions on earlier drafts on this paper. I assume full responsibility for any remaining vulnerabilities.

2.1.1 Stiglitz, Economics

p. A9

Free-rider: Someone who enjoys the benefit of a (public) good without paying for it; because it is difficult to preclude anyone from using pure public goods, those who benefit from the goods have an incentive to avoid paying for it, that is, to be a free-rider.

2.1.2 Justin Pappas Johnson 2001

Notes

Not many people are working on the same project at the same time.

Many people in the same project creates crowding and a need for coordination, this in turn makes participators create new projects within the existing project.

Modelling a world with many developers – the combined talent of the internet – is wrong projects are always relatively small. Interviews have confirmed this, when elaborating on some code it is checked if the code is in active development, which it seldom is.

Free riding is not possible if the wanted feature does not exist!!

2.1.3 Karin Knorr-Cetina 1982

In science there are large gains from incremental development. Incremental development is almost always intrinsically rewarding to the developer. Scientists analyse the benefit from contributing to some research. Large projects over large periods of time where the probability of success is difficult to calculate is not preferred. Smaller projects in existing fields are easier to predict and relatively small efforts may produce large gains.

This might also be the case for OSS, Many projects start out small and crude – but they are solving the problem. Any enhancement to such a project, often small, will be rewarding to the developer. But given the size of the patch and time required to develop the patch represent a relatively small cost to the developer.

3 The economics of OSS development and free riding

The basic idea: Is free riding possible in OSS. Some authors think so.

When a good is subject to public collective provision and it is not possible to exclude people from consuming the good, we should expect massive free riding.

Free-rider: Someone who enjoys the benefit of a (public) good without paying for it; because it is difficult to preclude anyone from using pure public goods, those who benefit from the goods have an incentive to avoid paying for it, that is, to be a free-rider.

Examples include professional fire fighting brigades. Some people subscribe to this service. Other people have houses surrounded by houses covered by the fire-fighting brigade. The house in the middle have no incentive to subscribe as the fire fighting brigade will surely arrive in case of fire – to prevent the fire from spreading to the subscribers houses.

Some provide free public goods if their personal benefit is sufficiently high. A ship owner might set up a lighthouse and buoys to make sure his ships make it safe to harbour. Other people might use the lighthouse and the buoys without paying for the service, and will be free riders.

These two cases are different because the first example shows that the free rider in case of fire will cost the fire brigade and thus the subscribers the cost of putting out the fire – however, this cost will be lower than the cost of the fire spreading. The free rider can expect the fire brigade to arrive given the mentioned cost structure. It is important to note that the fire fighting good is rivalrous in consumption, making the incentives to exclude high.

The second example shows an example where the owner of the lighthouse and buoys does not incur any cost from letting the other ships the equipment. These goods are non-rival in consumption making the incentives to exclude users low.

There are also two different situations to consider: 1) The good is existing, and 2) The good must be built/developed. In the following it is assumed that the goods are non-excludable but it is discussed whether the goods are rival or non-rival in consumption.

3.1 The good is existing.

If the good are non-rivalrous in consumption we are in a situation like the above lighthouse. Users have no incentive to provide the good since the good is existing and can be consumed in abundance – they are free riders. The provider of the good incurs no cost from the usage of the free riders. The provider may feel that there are and opportunity cost associated with their use. If the opportunity perceived cost is lower than the cost of making the good excludable so that the profit may be appropriated from providing the good - the provider have no incentive to exclude users. It might be that provision of the good is below social optimum since the provider only takes his personal needs into account when providing the good. For example, if the harbour was surrounded by reefs and fishing were particularly good it would be feasible to place buoys by the reef. The provider is, however, not a fisher and is ships are cargo ships. The provider will have no incentive to provide buoys at the reef, even though the fishermen might benefit from this. In this case deploying a few extra buoys would have been a relative small cost for the provider and a larger gain for the community. Because the provider only benefits from buoys that his ships use, these are the only buoys that will be deployed.

3.2 The good is not existing

If the good is not existing and must be built, the situation is inherently different. In this situation anyone can decide to provide the good. The provider will, however, incur a significant cost – the cost of providing the good. Assuming that the good is non-rivalous in consumption the cost of providing the good is occurring only once – neglecting maintenance for the sake of clarity. Using the lighthouse as an example the cost of building the lighthouse is significant. All users of the harbour will benefit from a lighthouse but the benefit varies greatly depending on the users needs. Some use the harbour for recreational sailing in the daytime only. The lighthouse is not necessary for their use but might be useful in some circumstances. The shipping company would receive a huge benefit from the lighthouse. Ships could come in at night and it would generally be much safer for ships to approach the harbour.

The users of the harbour could try to form a coalition and provide the good in common. However, this is very difficult as the users have different needs and their benefit from the lighthouse is also very different. Also each user knows that the other users desire the lighthouse and would benefit from building it. Thus the users might engage in strategic behaviour and wait for the others to build the lighthouse. They all know that once the lighthouse is build it is not possible to exclude the others from using the lighthouse, they also know that it is not possible to profit from the lighthouse other than the personal utility.

Now, the users of the harbour know that the shipping company is making a large profit from using the harbour. They also know that the shipping company would increase their profits if the lighthouse was built. So, the people using the harbour is maximising their utility by waiting for the shipping company to built the lighthouse. The users are behaving strategically and free riding.

3.3 How does this compare to OSS development?

The main point in strategic behaviour are the users ability to predict the supply of the good. In the case of building the lighthouse, the users could rationally predict that the shipping company would have interests in building a lighthouse. Also building of the lighthouse would require cooperation and agreements about the specific lighthouse, which would be difficult due to varying need. Importantly building a lighthouse requires a significant monetary investment which is greater that the economic ability of the individual users (in not they would have build the lighthouse as a private investment).

In case of the firefighters the free rider would also rationally expect that the good would be supplied.

Free riders are able to establish a need and to identify the supply of the good. This why they are able to rely on other as suppliers and free ride.

In open source software the situation is different. The only resources required are time and ability. In a way ability is a function of time spend, people learn when solving a problem. This means that anyone to some extent poses the resources required to solve a programming problem. There are no requirements of large organisations and vast up-front monetary investments.

In this situation time is the scarce resource. But what is also different is the way software is built. It important to recognise that any software can be built from an arbitrary number of small programs which in combination performs the actions of a single large program. There are many different gains from building large programs from many smaller. Smaller programs have the advantage of better maintainability – it's easier to gain a complete overview of the program and understand how it functions. This is referred to modularity, the software consists of many modules.

Modules, the smaller the better, has an advantage when new developers enter the project. It is much faster for new developers to understand individual modules than one large incomprehensible program. A new developer will quickly be able to target the module of interest and begin to understand its design.

Unlike the lighthouse, which must be built in one piece and where design changes in the middle of construction usually are costly, software has the ability to grow. Of course, there are design implications in software development. But, if the design has been made with extensibility in mind it is possible for the software to grow incrementally. There will be a core program that is able to interact with the different modules. Thus, new functionality can be added as a module at little cost.

This incremental approach to software development has implications for the organisation of software development. It is not necessary to build the complete and promised software at once. It is possible to implement just the most needed features and use those.

This means that is it no necessary to assign people to the different tasks in order to get a functioning program. I propose that waiting to assign people to tasks or not allowing others to participate in the development until the basic design has been made is actually very economic. By doing so the developer reduces the cost of communication and the cost of organisation. It is likely that given the opportunity people would made all kinds of suggestions if not committed to deliver part of the effort.

It is very difficult to collaborate in the initial phase of conceptualisation. I open source software this is especially true, since the means of communication are of limited bandwidth. In this case the first draft serves as basis for changes and discussion. Again the cost of communication and organisation are kept low. The software a communicated using the internet and the source code communicates all that has to be said about the program. The developers who made the first draft – the maintainer – does not have to present his program to others. The only communications that are conveyed with the program are most often a readme file that explains the intentions of the program and where the maintainer would like to see the program go. This and the source code are all the communication that goes from the maintainer to the developers. The developers create their own ideas of how the program should develop. They develop or fix the program to fit their needs, this is their motivation and the solution to their problem is their reward.

Returning to the cost of coordination: This is small or rather it is zero for the maintainer. He bears no cost for finding developers that can add to the program. The developers incur a cost when searching for software that can solve their problem and a cost from testing and examining the programs they have found.

4 The simple situation of development

Open source software development: The incentives of giving.

Scarce resources are: Time and knowledge.

A developer develops a crude version of some software that solves his personal problem. The developer has an economic gain from this software, it automates some of his tasks and thus provide more time for other activities. The cost of producing this software was the time devoted to the development.

The agent now faces a choice: 1) Keep the software to himself, 2) Sell the software under commercial terms or 3) Release the software as open source.

4.1 Keep the software to himself

By keeping the software to himself the software will still solve the problem but not get developed further. There are little incentive other than fun and personal interest to elaborate on the software and for instance make a nice intuitive user front end. This choice provides no gain and costs nothing.

4.2 Sell the software under commercial terms

Selling the software on commercial terms require that the developer makes the software easy to use and install. Also the developer must spend resources testing the software under different conditions. If the software does not reach product quality the developer must foresee costs arising from dissatisfied customers requiring support, improvements and possibly money refund. There are substantial costs related to making the software production quality not calculating the costs related to marketing. In this situation there is the possiblilty of negative response from the market, with lost investment as a consequence. This choice surely has related costs both monetary and time. There are large uncertainties regarding profit.

4.3 Release the software as open source

Releasing the software as open source costs nothing and users might provide feedback and contribute to the development. Contributions can be added features, comments and bug fixes. In addition a user group create a network of people with a similar interest: the software. Contributions from users in any form can be considered a profit derived from the work of the developer. Most contributions will contain some sort if insight on behalf of the user which the developer might find useful. Any features added or bug fixed are value added to the software, for which the developer incur no cost. When incorporating the added features og bugs fixed in the software and making a new releases the developer incur a cost: the time spend incorporating the changes. However the time spend incorporating changes is less that the time the users spend developing the added features or fixing the bugs that were causing them problems. As such, the developer receives a net gain from incorporating the changes there is a net gain. However, the developer might not find the suggested changes interesting and thus the developer has the choice to reject and not incorporate the suggested changes. When rejecting changes the developer risk that the contributing user no longer will contribute to the software. It is not possible for the developer to estimate the potential usefulness of a user that contributes. However, of the number of people using software the number of people who contributes back are relatively small. This makes the value of users that contribute generally high. Therefore a developer is often inclined to treat contributing users very well and try to make them stay connected to the project instead of rejecting their suggestions. When releasing the software the developer incur no cost, the developer incur a potential cost equal the value of the software if released as commercial software. As noted in comment nr.2 there are surely costs related to releasing the software as a commercial product but guarantee of profit. This makes it likely that small projects intended to solve personal problems will be released as open source software. The costs related to maintaining the software i.e. incorporating changes from users are relatively small compared to the cost the users incur when spending time developing the changes. I.e. there is, time wise, a net profit from accepting contributions and incorporating those.

This was the example of simple software development or rather the beginning of a software project. Let us now consider an existing project, with a defined user base and dedicated maintainers. What are the incentives for users to turn contributor and spend time developing new features for the software. The perspective is on the user that has the possibility of contributing to the software.

A user receives a piece of open source software and begins to use the software. The users expectations to the software might be different from that of the original developer also the configuration in which the software is used might also differ from that of the developer. This leads to the possibility that the user does not become satisfied with the software. Either the software does not have the expected features or the software is buggy and the wanted features is not functioning as desired.

Assuming that the user downloaded the software in the first place because he needed some particular feature or functionality. The user now has a choice: 1) Discard the software, and 2) elaborate on the software to fit his needs.

4.4 Discard the software

1) If the user chooses to discard the software the user still has an unsatisfied need. In this situation the user can search for other software that might fulfil this need, buy the software assuming the software exists or make the software himself. Lastly the user may decide to postpone his need and wait until the wanted software becomes available.

4.5 Elaborate on the software to fit his needs

5 Conclusion