

Contract no. 030776	Workpackage WP1	Delivery D1.5	Delivery Date 2007-08-31
------------------------	--------------------	------------------	-----------------------------

CASCADOSS

Development of a trans-national cascade training programme on
Open Source GIS&RS Software for environmental applications

SPECIFIC SUPPORT ACTION

PRIORITY 1.2.4.2.2: IDENTIFICATION OF NEW METHODS OF PROMOTING AND
ENCOURAGING TRANSNATIONAL TECHNOLOGY TRANSFER

DELIVERABLE 1.5

Inventory and analysis of OSS business models

Start date of project: 2007-05-01

Duration: 24M

Leading contractor: KULeuven

Revision: final

Table of Contents

1	Introduction	6
2	What is F/OSS?	7
2.1	Defining F/OSS.....	7
2.2	F/OSS and Intellectual Property	7
2.3	F/OSS and reciprocal licenses	7
2.4	F/OSS as a software development model	8
2.4.1	Community-driven development	8
2.4.2	Open development and Informational density	10
2.4.3	“Forkability”	11
2.4.4	Meritocratic development.....	11
2.5	F/OSS as a software distribution model	12
2.6	F/OSS as a business model	12
3	The Economics of F/OSS	14
3.1	The Motivational Puzzle	14
3.2	The private provision of public goods	15
4	A Descriptive framework for F/OSS Business Models.....	18
4.1	Components of a Business Model.....	18
4.2	The Architecture of Value Creation	18
4.2.1	The Market Design	18
4.2.2	The Software Value-Chain.....	19
4.2.3	The Distribution Model.....	20
5	F/OSS Business Models	21
5.1	Business Model Overview	21
5.2	The Dual-licensing Model	21
5.2.1	The Value Proposition	21
5.2.2	Product/services	21
5.2.3	Architecture of Value Creation.....	22
5.2.3.1	Market Design	22
5.2.3.2	Software value chain.....	22
5.2.3.3	Distribution model	22
5.2.4	The Revenue Model	22
5.2.5	The role of F/OSS licensing in the model	22
5.2.6	Examples of companies that adopt the model.....	23
5.2.7	Strengths/weaknesses of the model.....	23
5.3	The Support Seller Model.....	23
5.3.1	The Value Proposition	23
5.3.2	Product/services	23

5.3.3	Architecture of Value Creation.....	23
5.3.3.1	Market Design.....	23
5.3.3.2	Software value chain.....	23
5.3.3.3	Distribution model.....	24
5.3.4	The Revenue Model.....	24
5.3.5	The role of F/OSS licensing in the model.....	24
5.3.6	Examples of companies that adopt the model.....	24
5.3.7	Strengths/weaknesses of the model.....	24
5.4	The Third-party Support Seller Model.....	25
5.4.1	The role of F/OSS licensing in the model.....	25
5.4.2	Examples of companies that adopt the model.....	25
5.4.3	Strengths/weaknesses of the model.....	25
5.5	The Platform Provider Model.....	26
5.5.1	The Value Proposition.....	26
5.5.2	Product/services.....	26
5.5.3	Architecture of Value Creation.....	26
5.5.3.1	Market Design.....	26
5.5.3.2	Software value chain.....	26
5.5.3.3	Distribution model.....	27
5.5.4	The Revenue Model.....	27
5.5.5	The role of F/OSS licensing in the model.....	27
5.5.6	Examples of companies that adopt the model.....	27
5.5.7	Strengths/weaknesses of the model.....	27
5.6	The Consulting Model.....	28
5.6.1	The Value Proposition.....	28
5.6.2	Product/services.....	28
5.6.3	Architecture of Value Creation.....	28
5.6.3.1	Market Design.....	28
5.6.3.2	Software value chain.....	28
5.6.3.3	Distribution model.....	28
5.6.4	The Revenue Model.....	28
5.6.5	The role of F/OSS licensing in the model.....	29
5.6.6	Examples of companies that adopt the model.....	29
5.6.7	Strengths/weaknesses of the model.....	29
5.7	The Software-as-a-Service Model.....	29

5.7.1	The Value Proposition	29
5.7.2	Product/services	29
5.7.3	Architecture of Value Creation.....	29
5.7.3.1	Market Design.....	29
5.7.3.2	Software value chain.....	30
5.7.3.3	Distribution model	30
5.7.4	The Revenue Model	30
5.7.5	The role of F/OSS licensing in the model	30
5.7.6	Examples of companies that adopt the model.....	30
5.7.7	Strengths/weaknesses of the model.....	30
5.8	The Added-Value Provider Model	31
5.8.1	The Value Proposition	31
5.8.2	Product/services	31
5.8.3	Architecture of Value Creation.....	31
5.8.3.1	Market Design.....	31
5.8.3.2	Software value chain.....	31
5.8.3.3	Distribution model	31
5.8.4	The Revenue Model	31
5.8.5	The role of F/OSS licensing in the model	32
5.8.6	Examples of companies that adopt the model.....	32
5.8.7	Strengths/weaknesses of the model.....	32
5.9	The Accessorizing Model	32
5.9.1	The Value Proposition	32
5.9.2	Product/services	32
5.9.3	Architecture of Value Creation.....	32
5.9.3.1	Market Design.....	32
5.9.3.2	Software value chain.....	33
5.9.3.3	Distribution model	33
5.9.4	The Revenue Model	33
5.9.5	The role of F/OSS licensing in the model	33
5.9.6	Examples of companies that adopt the model.....	33
5.9.7	Strengths/weaknesses of the model.....	33
5.10	The Loss Leader Model	33
5.10.1	The Value Proposition	34
5.10.2	Product/services	34

5.10.3	Architecture of Value Creation.....	34
5.10.3.1	Market Design.....	34
5.10.3.2	Software value chain	34
5.10.3.3	Distribution model.....	34
5.10.4	The Revenue Model	34
5.10.5	The role of F/OSS licensing in the model	35
5.10.6	Examples of companies that adopt the model.....	35
5.10.7	Strengths/weaknesses of the model.....	35
5.11	The Widget Frosting Model.....	36
5.11.1	The Value Proposition	36
5.11.2	Product/services	36
5.11.3	Architecture of Value Creation.....	36
5.11.3.1	Market Design.....	36
5.11.3.2	Software value chain	36
5.11.3.3	Distribution model.....	36
5.11.4	The Revenue Model	36
5.11.5	The role of F/OSS licensing in the model	37
5.11.6	Examples of companies that adopt the model.....	37
5.11.7	Strengths/weaknesses of the model.....	37
6	The Effects of F/OSS on the software market	38
6.1	Threat of new Entries	38
6.2	Bargaining Power of Suppliers	39
6.3	Bargaining Power of buyers	39
6.4	Rivalry among Existing Firms	39
6.5	Threat of substitutes	39
7	Open Source Business Risks	40
7.1	License compliance	40
7.2	Liability.....	40
7.3	Patents	40
8	Open Source Business Models in Practice.....	41
9	Application to the geospatial market.....	42

1 Introduction

This report presents an overview and analysis of the business models for Free/Open Source Software (F/OSS). The analysis will focus on the question how open source creates interesting commercial opportunities for companies. The key differentiator of F/OSS is that its source code is open and freely available. Although this implies that F/OSS producers cannot generate revenue from licensing, it confers benefits to both software producers and consumers and thereby creates new commercial opportunities. What these opportunities are, and how they relate to the distinctive features of F/OSS, will be examined in this report with a special focus on the geospatial communities in Europe.

The primary intent of this report is to educate and inform about F/OSS; not to further the state of knowledge. It is therefore an attempt to synthesise the insights of the relevant scientific, technical and business literature. The synthesis aims at a coherent overview of what open source means for businesses.

2 What is F/OSS?

2.1 Defining F/OSS

The Free Software Foundation (FSF) defines free software succinctly in terms of “the users' freedom to run, copy, distribute, study, change and improve the software” (<http://www.fsf.org/licensing/essays/free-sw.html>). The Open Source Initiative (OSI) maintains a public open source definition at <http://www.opensource.org/docs/osd>. The OSI definition is the most widely used definition of Open Source, and has received strong support in the developer community. For our purposes the differences between the OSI and FSF definitions are minor. Both definitions agree in essential points about the rights that the author of the software must grant licensees (users) in order to qualify as F/OSS. Rosen enumerates these rights as follows (Rosen 2004):

1. Licensees are free to use open source software for any purpose whatsoever.
2. Licensees are free to make copies of open source software and to distribute them without payment of royalties to a licensor
3. Licensees are free to create derivative works of open source software and to distribute them without payment of royalties to a licensor
4. Licensees are free to access and use the source code of open source software
5. Licensees are free to combine open source and other software

Rosen's enumeration is compatible with both the FSF and OSI definitions of F/OSS.

The fact that the creator of software grants these rights to its users accounts for the special characteristics of F/OSS as a development and distribution model and as a foundation for business models.

2.2 F/OSS and Intellectual Property

F/OSS is not public domain software. Public domain software is the property of no one, and hence no one can impose any restrictions on how it is used and what is done with it. Most F/OSS software, however, is not public domain but owned by individuals and companies. By means of their copyright, the authors can and typically do impose requirements on licensees of the software. There are limits to what one can do with F/OSS source code. For instance, most licenses usually prohibit to simply copy-and-paste parts of code without attribution.

The distinctive characteristics of F/OSS are due to the fact that the software freedoms are granted as a license over copyright. A F/OSS license grants broad usage and redistribution rights to the licensee. These rights are so broad that they come to resemble ownership rights. The licensee doesn't really own, or hold the copyright to, the source code, but he has the right to use, modify and re-distribute it – rights traditionally reserved exclusively for the owner (Scott 2006). Equally important, however, F/OSS licenses can also impose certain legally binding requirements. These requirements enable F/OSS developers to protect the intellectual integrity of their work (e.g. attribution notices), make “copylefting” possible, and protect contributors from appropriation of their work.

2.3 F/OSS and reciprocal licenses

Some F/OSS licenses impose the requirement that software that uses or is based on the F/OSS software must also be distributed under a F/OSS license. This is the most notable feature of the GPL license – one of the most popular open source licenses. F/OSS licenses with this characteristic are called copyleft (Stallman 1984), strong (Scott 2006), or reciprocal

licenses (Rosen 2004). As Richard Stallman explains, “copylefting is a general method for making a program or other work free, and *requiring all modified and extended versions of the program to be free as well*” (Stallman 1984, our emphasis). The GPL (GNU General Public License), the MPL (Mozilla Public License), and the CPL (Common Public License) are popular reciprocal licenses.

F/OSS licenses which lack this feature are called weak (Scott 2006) or academic licenses (Rosen 2004). Examples of academic licenses are the BSD, Apache and Artistic licenses.

The difference between academic and reciprocal licenses reflects a fundamental difference of opinion on the value of F/OSS. Proponents of academic licenses such as the BSD and Apache communities want to create a public commons of software. Other developers can use this software commons as a resource, but they are not required to contribute something back to that commons. Academic licenses allow software from the commons to be used to create proprietary software. Reciprocal licenses, however, mandate that those who derive work from software in the commons place the derived work back into that commons.

The Lesser GPL (LGPL) License takes an interesting intermediate position between the reciprocal and the academic licenses. The license itself is reciprocal in the sense that any modification of the source code must be made public if the modified software is distributed. In this way, the LGPL guarantees that improvements to the software remain within the open source commons. But if the software is used as a component or library within a larger system, the license does not impose any constraint on how the larger system is licensed. The LGPL functions as a reciprocal license when the software is extended or modified; it functions as an academic when the software is used as a part in a larger system.

2.4 F/OSS as a software development model

The software freedoms enumerated in section 2.1 have a huge effect on how software is developed. In the next sections, we discuss several features of the F/OSS development model that make it so powerful.

2.4.1 Community-driven development

Since the source code is open to all, there is no boundary that divides the users and developers into those with access to the source code and implementation internals of the software and those from whom this information is kept secret. The absence of such a boundary empowers the user community to become more actively involved in the software development process. The community of a F/OSS project therefore typically comprises the full spectrum from ordinary users, to user-contributors (who contribute by testing, feedback, bug-reporting and documentation), to developers (who write the source code).

Since the community around a F/OSS project is so empowered, the core developer group can change organically over time. Users and/or other developers may want to get actively involved in the actual code writing for a variety of reasons (to have their favourite feature implemented, to remove bugs that bother them, or just to have something interesting to do on a Sunday afternoon). If they find that the license that covers the F/OSS project protects their interests sufficiently, they may volunteer to co-operate with the initial developer or developers. (Of course, it is up to the original developers whether or not to accept them. Most mature F/OSS projects have some sort of process that volunteers have to go through before they are accepted as co-developers).

If the core developer group wants to attract contributors and developers, not only the source code (product) also the development (process) must become open and transparent (Fogel 2006, pp.36ff). This means that most F/OSS project are open about a lot more than the source code: design documents, technical discussions on design and implementation issues, etc. are public and freely available to users and developers alike.

In his seminal essay *The Cathedral and the Bazaar*, Eric Raymond describes several lessons from successful F/OSS projects. Several of these lessons relate directly to the existence of a community that is empowered to take an active role in the development process (Raymond 1998). We enumerate these here.

“Every good work of software starts by scratching a developer’s personal itch”

Many F/OSS projects start from a problem that a user-developer has. Such projects are therefore not market- or technology-driven but user-driven: they start from a genuine user need that cannot be adequately satisfied by existing offerings in the software market (at a price acceptable to that user). User-developers may in such circumstances decide to initiate a F/OSS project. F/OSS projects with such lineage are often far better attuned to the real needs of the user-community than many commercial competitors.

“The next best thing to having good ideas is recognizing good ideas from your users. Sometimes the latter is better.”

Users/developers who care about a particular F/OSS project may contribute by suggesting ideas for improvement from their own experience. E.g. the GIS world counts many sophisticated and knowledgeable users whose insights would be valuable in steering the development of GIS systems. By volunteering their insights through e-mail lists and discussion forums, they may influence the development of the software. Not only that, because the source code is available to them and much development information is readily accessible, they can also verify to what extent these contributions have been taken up by the developers. This type of verification is very encouraging and motivating to users, but very hard to do in closed source development models,

“Treating your users as co-developers is your least-hassle route to rapid code improvement and effective debugging”

“Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone.” or (“Given enough eyeballs, all bugs are shallow”).

Both these lessons derive from the fact that some (if only a minority) of users become sufficiently knowledgeable about the internals of the software so that they start contributing code and fixes to the project. Users that are sufficiently motivated to develop a new feature can do so and submit the code to the project developers. If the developers find it is a valuable addition to the code base, they will incorporate it. In this way, the software can gain features that were not anticipated (nor planned) by the project developers. Very often, such a feature is something that the user developed for his own use and decided to give back to the community. Even more important and more common, are users that in the course of their use of the software need to investigate problems and bugs. Since they have access to the source code the more technically-sophisticated users will often use the code to examine and identify the problem. This leads to much more specific problem descriptions, and suggestions for fixing the problem.

The community that surrounds a successful F/OSS project is not homogeneous, of course. And not all users are sufficiently capable or interested to become actively involved in a F/OSS project. Mockus et al. (2000) studied the Apache web server development process and the contributions made by the community. Their findings for the Apache Server are:

- 10-15 core developers wrote more than 80% of the code.
- A group larger by an order of magnitude than the core will repair defects, and a group larger by another order of magnitude will report problems.

To put these data in context: at the time of writing Mockus et al. estimated the number of Apache Server installations at around half a million. The total number of individuals that

submitted problem reports numbered around 3000. This is less than 1% of the number of installations.

These findings and research by others show that the community that surrounds a F/OSS project is highly stratified (Gosh and Prakash 2000; Gosh et al. 2000; Hunt and Johnsson 2002, Healy and Shussman 2003). A small group of core developers, is surrounded by a larger group of contributors/testers. In a second layer are active users that submit bug reports, and respond to user's questions on forums and mailing lists. By far the greatest group are users that remain passive consumers of the software. Our experience with projects in the geospatial domain, such as JTS, PostGIS, GeoTools and GRASS support these findings.

As the above shows, it is only a rather small part of the total user community that is actively engaged in the project. Nevertheless, numerically this is still quite a lot more than are typically engaged in comparable proprietary software development. Equally important is the fact that contributors come from various backgrounds, both technically and culturally. Combined with the meritocracy that is often evident in F/OSS development (see below), this is an advantage because ideas and implementations will need to survive examination and peer-reviewed from a wide variety of viewpoints. Lastly, most contributors and core developers self-select for the development tasks based on their capabilities, know-how and interest. E.g., most contributors to GeoTools and PostGIS seem to be capable software developers interested in GIS for professional reasons. This leads to high levels of motivation and commitment.

2.4.2 Open development and Informational density

One important advantage of the F/OSS development model is that it allows outside contributions (contributions from outside the original core developer group). But, as already remarked, this requires a fully transparent, well-documented open development process. The reason for this becomes obvious when we consider what is involved in submitting a first contribution to a project:

1. The contributor must set up a suitable build environment.
2. He must create, and then test his contribution.
3. He must convince the original developer group to review his work, and accept it into the code base.

If the development process is not open or transparent, potential contributors will face considerable problems in steps 1 and 3. First, setting up a build environment can be quite complex. If the development process is not well-documented, then information on setting up a proper build environment will typically be outdated or incomplete (the original developers already have their own build environments and do not need this information). Even more problematic is step 3. In case the development process is not open, the potential contributor has no guarantee that his contribution will be accepted. It might be because the contribution may not fit the road plan, because the contribution doesn't satisfy some guideline that he is unaware of, or simply because the original developers aren't interested. Problems at step 3 are the most pernicious, because they will quickly undermine the project's credibility as a genuinely open, community-driven project.

Projects that are insufficiently open will miss out on outside contributions because potential contributors will feel that they have little chances of having their contribution accepted at least within a reasonable amount of time and effort. So, if a project wants to attract outside contributions or new developers, it must minimally:

1. Clearly document the code development processes (build tools, testing, coding style, ...).

2. Clearly articulate the project's mission statement, vision, development philosophy and road map.
3. Discuss publicly the road map, status and progress of the project.

The flip-side is that projects that have this completely transparent open development process have good project documentation. Everything that a new-comer needs to get started with development, and to understand the goals and direction of the project, are already well-documented and easily accessible. This level of documentation is sometimes very hard to achieve in a closed-source project.

Although we find that truly open F/OSS projects are normally quite well documented from a development point of view, the documentation does not consist of a set of documents. Rather it is scattered in mailing lists, wiki pages, issue tracking systems, and even in the source code.

2.4.3 “Forkability”

One implication of the freedoms F/OSS grants is the freedom to fork the project. A group of developers can fork a project by copying the code and starting a competing project around the copy. Well-known examples are the forking of Emacs into Emacs and Xemacs, XFree86 into XFree86 and X.org. An example from the GIS world, is the forking of OpenJump from the original Jump project.

Forking is generally considered a “bad thing”: it entails duplication of work, splits development and user communities and can result in infighting in the F/OSS community. Forking also generally imposes a burden on the F/OSS users. They must make a choice to remain with the original project, which might be severely hampered by the defection of a part of its community, or move to the forked version, which might die out by lack of resources or interest. The forkability of F/OSS therefore introduces business risks for its users.

Nevertheless, the risk of forking also means that F/OSS project leaders have an important incentive to be responsive to the concerns of their users, contributors and developers. If this is not the case, community members can decide to fork, and start with a new project. As Fogel (2006, p.88) says: “Replicability implies forkability; forkability implies consensus.” In this way, forkability functions as a safeguard, much like customer satisfaction does in the commercial marketplace.

In the case the original project carries a reciprocal license, forking is also relatively “benign”. The original license must be carried over to the new license, and all changes and improvements made to the fork are also available to the original. This helps ensure that forking will only happen for very good and weighty reasons and not for purely commercial or ego-driven reasons.

2.4.4 Meritocratic development

An important difference between open and closed software development processes, lies in the project management and leadership. As already indicated in our discussion of forkability, the leader of a F/OSS project have no absolute control over the project. They must set up a project governance structure that typically includes a mission statement, a conflict resolution procedure, and a procedure by which new developers can be added to the core development team (developers with “commit” access to the project repositories). To be workable and sustainable a F/OSS project needs to construct a governance structure that is fair and motivating to those who put the most effort in it. That means that most F/OSS projects leadership is meritocratic in nature: those who make valuable contributions will have a say in the direction of the project proportional to the magnitude of these contributions. Needless to say that such meritocratic leadership is highly motivating for good developers. It is also a good guarantee that technical excellence and/or end-user requirements drive the development road map and not commercial considerations.

2.5 F/OSS as a software distribution model

F/OSS is also software distribution model (Olsen 2006). F/OSS allows the widest possible distribution of a software package, at very low cost and with minimal marketing. In fact, some companies adopt an F/OSS business model for this reason alone and do not adopt a F/OSS development model (or only to a minimal degree).

Here are some of the reasons why F/OSS typically gets widely distributed:

- Because all F/OSS licenses allow the re-distribution in source or binary form, the software may be distributed by agents other than the F/OSS project owners. The most widely known example of this is the Linux kernel. This large market share for Linux in the server market is probably due in large part to the fact that it was distributed by a large range of firms and organisations (Debian, Slackware, Mandrake, Red Hat, SUSE, Gentoo, IBM, ...).
- Since F/OSS software can be downloaded and tested without monetary costs, without the need to contact vendors by phone or e-mail and without time limits, it is far more easy for prospective users to test and explore the software.
- Since the source code is open prospective users can better gauge the quality of the software. This is especially so when the software is developed according to an open source development model. For example, in open F/OSS development models the bug lists are open to view to anyone, allowing prospective users to see how many bugs there are, how quickly they are resolved and how professionally. This openness fosters confidence in the software. Even if the result of the examination is mixed, the prospective users has an accurate view of what they get. This is usually not the case with proprietary systems where you know what you get only after spending large amounts of money and time.
- Many developers like working with F/OSS during software development because of its low cost, – even when the developed software will run on proprietary software. Often, this spills over to production environments: having gained sufficient trust in these F/OSS components, they will start proposing these components for production use. (E.g. Java web applications are often developed using the F/OSS Tomcat application server and then deployed in production on BEA WebLogic or IBM WebSphere application servers).
- When software developers need to select a library or component they often prefer F/OSS because these are not completely black-box. The availability of the source code means that they can trace bugs and problems through to the source code of the library or component (e.g. in a debugger).

2.6 F/OSS as a business model

When F/OSS is essential for a business model we speak about open source business models. If we look at the software value-chain (see §4.2.2), it is clear that a F/OSS license effectively prevents a developer from directly appropriating the value of his software development effort. Other revenue-generating activities in the value chain such as training, support and consultancy remain unaffected. In that sense, F/OSS business models are not a complete departure from traditional models (see also Young (1999) and Behlendorf (1999) concerning the similarities between F/OSS and traditional software business models).

Nevertheless, making money directly from the software product is traditionally held as the key for a successful software business model. Software development is characterized by high fixed costs and virtually zero marginal costs. In a closed source model, a company can generate revenue from licensing, and so generate very high margins of return provided the customer

base is large enough to recuperate the fixed costs. Here is how Olsen (2006) summarises the importance of licensing revenue:

Software licensing revenue is good revenue – because you can make and license a second copy of a piece of software for essentially no additional cost, business and the financial markets like licensing revenue.

The question for F/OSS business models is whether they can compensate adequately for this lack of licensing revenue. This is the subject of the rest of this report.

3 The Economics of F/OSS

F/OSS is a relatively new mode of software production. In this section we will briefly discuss its economic underpinnings. We will argue that there are compelling arguments for believing that the F/OSS mode of software production is sustainable and provides tangible economic benefits. We believe that such a discussion provides a necessary background for understanding the F/OSS business models. In the academic economic literature, F/OSS presents several puzzles. First, there is the *motivational puzzle*. Why do developers work on software whose value they cannot appropriate for themselves? The F/OSS licensing scheme makes software non-excludable and non-rivalrous in consumption. In effect that means that F/OSS can be considered a public good. This leads to a second question: *Why do private actors collaborate to create a public good, and how do they deal with free-riding?*

3.1 The Motivational Puzzle

The motivational puzzle is: Why do F/OSS developers spend so much time and effort on a product that they essentially give away for free? Assuming a traditional economic perspective, this is only rational behavior if the benefits of doing this exceed the costs for the developer. The question then becomes, what are the benefits that individual developers gain? Lakhani and Wolf (2005) distinguish between two types of motivations: intrinsic and extrinsic motivations. A motivation is intrinsic when it is inseparable from the activity itself. Aspects of the activity itself motivate the developer. An extrinsic motivation is a separable consequence of the activity. Lakhani and Wolf subdivide intrinsic motivations into enjoyment-based and obligation/community-based motivations. As regards the extrinsic motivations, they take up the proposal by Lerner and Tirole (2002) that extrinsic benefits consist of immediate and delayed payoffs. Among the immediate benefits are:

- being paid by a company to work on F/OSS projects (e.g. Red Hat or IBM engineers working on Linux)

- being a direct user, and thus beneficiary, of the F/OSS that the developer works on

Among the delayed benefits are:

- career advancement (job market signalling)

- improvement in skills through collaboration and peer review

Lakhani and Wolf surveyed individual developers of F/OSS projects hosted on sourceforge.net. Their study started in 2001. The most important reasons that developers cite for their participation are that they need the software product they are working on (58%), and the intellectual stimulation provided by F/OSS development (45%). The third most important motivator is the opportunity to improve one's software development skills. The belief that software should be open is also an important community-based motivator (33%). In light of these results, Lakhani and Wolf conclude that "there is no single dominant explanation for an individual software developer's decision to participate in and contribute to a F/OSS project" (p.19).

When the authors examine the factors that determine the level of effort the developers spend on F/OSS, they found that the only significant determinants are (in order of importance):

- Enjoyment-related intrinsic motivations in the form of a sense of creativity

- Extrinsic motivations in the form of payment

- Obligation/community-related intrinsic motivations.

Empirical research on intrinsic motivations finds that in general extrinsic benefits such as being paid have a negative impact on intrinsic motivations. Lakhani and Wolf find, surprisingly, that this negative relationship is absent in F/OSS. For the large group of developers that are being paid for their participation (40% of the sample), the fact that they are being paid does not diminish the sense of creativity and/or community-involvement.

3.2 The private provision of public goods

In the Lakhani and Wolf study of 2001 about 40% of the F/OSS developers were being paid for their contributions. Considering the increased involvement of companies such as Red Hat, IBM, and Novell in the F/OSS ecology since then, it is safe to assume that the relative size of this group has grown substantially in the last couple of years. This points us to our next question: *Why do private firms collaborate to create a public good, and how do they deal with the free-riding problem?*

Companies engage in F/OSS either by making developer resources available to a community-based project, or by setting up F/OSS projects themselves. The first case is exemplified by Red Hat that contributes heavily to the GCC (GNU Compiler Collection – the most important set of compilers for GNU/Linux). A good example of the latter case is Maemo. Maemo is a Linux-based F/OSS development platform for the Nokia tablet PC. Maemo was set up by Nokia with the express intent of promoting a developer community able to develop high-quality applications for the Nokia Tablet.

Companies are motivated to do this mainly for extrinsic motivations. Either they see an interesting business opportunity in F/OSS and adopt a F/OSS-based business model, or they do it for cost-efficiency reasons, or they do it for tactical and strategic reasons. The first group is exemplified by Red Hat or Novell that have adopted F/OSS based business models. These models will be discussed extensively in §5. The second group is typified by system integrators. System integrators like F/OSS because it enables them to increase profits through direct cost savings. The third group, those motivated to contribute by strategic or tactical reasons, is exemplified by the Maemo project discussed above.

Recent research suggests that some companies using F/OSS are also *intrinsically* motivated to devote resources to F/OSS (Rossi and Bonaccorsi, 2006). In this context, intrinsic motivation consists in adherence to F/OSS community values. These firms support F/OSS from a sense of obligation to give back to the community. As can be expected, Rossi and Bonaccorsi found that this holds for a respectable minority of firms (35%). The authors believe that these companies “have probably inherited their community-oriented attitudes from founders that were previously involved in [F/OSS] programming at the individual level and have turned their passion into a profession.” (p.105).

Despite the existence of these intrinsically motivated companies, it is clear that most companies engage in F/OSS for opportunistic and self-centred reasons. Many observers fear that these companies will “take out more than they put in”. More precisely, they fear that companies will try to appropriate as much value as possible from a F/OSS project (e.g. by offering complementary services), while committing as few resources as possible to the project. This is a case of the “free-rider problem”. Here is how Wikipedia describes it: “free riders are actors who consume more than their fair share of a resource, or shoulder less than a fair share of the costs of its production. The free rider problem is the question of how to prevent free riding from taking place, or at least limit its negative effects. (source: http://en.wikipedia.org/wiki/Free_rider_problem).

In general, free-riding behavior is seen as destructive because it tends to undermine the motivation of those actors that do take up their fair share of the burden. Recent research, however, suggests that F/OSS is relatively insensitive to free-riding behavior. Rossi and Bonaccorsi, in a study on the contributions made by firms to F/OSS find that “open Source communities permit some member to take much more than they give, provided they do not violate minimal membership rules. ... The literature on [Common Property Regimes], public good provisioning and free riding has probably overestimated the potential destructive role of a

small number of non contributors, assuming that their behaviour should inevitably self-propagate. This is not necessarily true” (Rossi and Bonaccorsi, 2004).

We believe that the relative insensitivity of F/OSS to free-riding is the consequence of several features of F/OSS. The first, and perhaps most important is that F/OSS presents to many companies a third option to the traditional “make-or-buy” dilemma (Bessen 2006). Companies that have highly specific needs, can collaborate with other companies and individual developers on a “platform” that can then easily be customized for their own specific purposes. A very prominent example of this is Linux. Linux is used as operating system by mobile phone (Motorola), consumer electronics (TiVO), and computer equipment vendors (IBM). The modularity of Linux makes this broad array of uses possible. It provides a powerful and stable “platform” that serves as a point of departure for the custom developments that these companies require. Because the collaborating companies benefit from a high-quality base product, they are motivated to co-operate to ensure the continued existence and quality of the base product. Free-riding is not an option for these companies because they have highly specialised needs: no other company or set of individuals is likely to produce exactly the product that they need. Baldwin and Clark (2003) also argue that free-riding in F/OSS projects will be limited if the product is highly modular. By “modularity” the authors mean that individual features can be added to the software base product easily, incrementally and independently from other project participants. Highly modular software with a very large potential feature set lowers the cost to develop an additional feature, while making free-riding behavior unattractive: because a relatively small group of developers can select features to implement from a large set of potential features, free-riders face a low probability that a developers implements the feature the free-rider wants.

The argument that modularity limits the damage from free-riding is strengthened by two other aspects of software in general and F/OSS in particular. One is that “platform” or “infrastructure” code is actually not necessarily very resource-intensive. For instance, in an audio interview, Mark Shuttleworth, the multi-millionaire founder of the popular Ubuntu Linux distribution, commented that he could keep the Ubuntu project funded for “a couple of decades if necessary” out of his own pocket. Something similar holds for the Apache webserver. The study of Mockus, Fielding and Herbsleb on the Apache project found that a mere 15 core developers contributed more than 80% of the source code (2000). These relatively small teams are able to keep the platform in good shape so that others can contribute small, incremental features that collectively increase the value of the total product. This also suggests that the cost of contributing on the platform is for many companies sufficiently small relative to the value of the complete product that most companies will not free-ride: the potential bad press and bad community relations are simply not worth the cost-saving of free-riding.

The second feature that makes F/OSS resistant to the damage of free-riding is that software in general is characterized by strong positive network effects. The value of a piece of software increases as the number of users of the software increases. In the case of F/OSS, companies that free-ride are still increasing the uptake and usage of the software. The increase in popularity of the product will usually imply an increased motivation for the individual developers (for intrinsic reasons such as pride in the result, but also extrinsic reasons, signalling one's programming abilities to a wider audience, e.g.). So the negative effects of free-riding are dampened by positive network effects.

In case the F/OSS is covered by a reciprocal license such as the GPL, the licence itself lessens the damage of free-riding. Although free-riders profit from the fruit of someone else's work, they cannot take it out of the commons and turn it into a commercial product. So free-riders cannot do very much damage because they cannot create a rival product to the original.

A final reason why free-riding is not so damaging in F/OSS, is that knowledge is an intangible asset. The companies and individual developers that contribute to F/OSS has a depth of knowledge about the product that free-riders have not. Although, in principle everything is open to view and available to anyone, in practice in-depth knowledge can only be acquired by working on the code and start developing in it. In fact, customers of F/OSS services know this

also, and therefore will prefer to acquire services from a company that has made genuine contributions to the F/OSS rather than from one of its free-riding competitors.

This is not to say that free-riding provides no problems for F/OSS. Significant free-riding can diminish the chances of success of a F/OSS project. But it does seem that free-riding is less of a problem than in other domains.

4 A Descriptive framework for F/OSS Business Models

4.1 Components of a Business Model

For the purposes of this discussion, we adopt a simplified conceptualisation of the structure of a business model that is based on Osterwalder (2004) and Stähler (2002). According to this conceptualisation, a business model consists of the following components (Osterwalder 2004, p. 31).

Component	Questions to ask
Value Proposition	What value does the company create for customers and partners?
Product/services	What does the firm sell?
Architecture of Value Creation	How and through what configuration is value created? Where in the value-chain is the company positioned?
Revenue Model	How does the company earn money?

4.2 The Architecture of Value Creation

In our discussion of F/OSS business models we will specifically highlight the distinctive role of F/OSS in the architecture component. We simplify Stähler's description of this component to the following elements: the market design, the position in the software value chain and the distribution model (interface between the company and its customers).

4.2.1 The Market Design

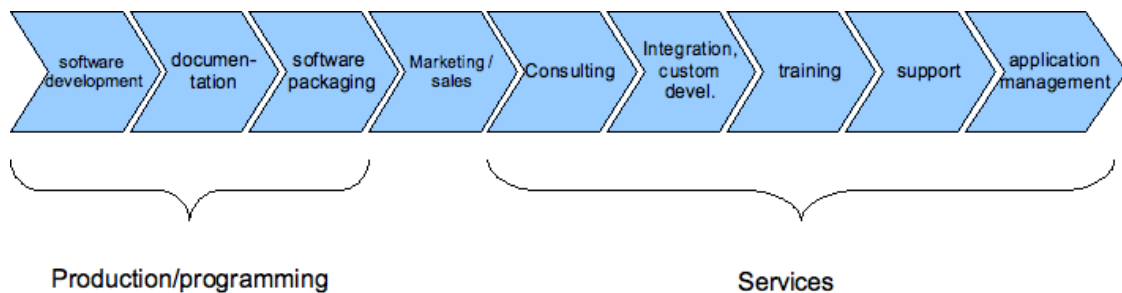
The market design specifies the markets that the company targets in the business model. These markets can be segmented based on geographic location, type (Business-to-Business or B2B, Business-to-Consumer or B2C), or whether it is private or public sector. We propose the following segmentations:

- geographical segmentation:
 - Local market: customer's are geographically near and/or share a cultural or linguistic background with the company.
 - Global market: the geographic location of the customer is largely irrelevant to the business model.
- Type segmentation:
 - B2C – business-to-consumer: the customers targeted in the business model are end-user of the product or service offered
 - B2B – business-to-business: the customers are other businesses that use the product or service as an input for their own offerings.
 - B2G – business-to-government: the customer is a public sector body.
- Software product segmentation (Berlecon Research 2002):

- Enterprise solutions: The model targets customers interested in solutions that support his business processes (Customer Relationship Management, Enterprise Resource Planning)
- Packaged Mass-market software: the model targets customers interested application software that satisfies highly general computing needs such as text processing, image processing, or spreadsheeting.

4.2.2 The Software Value-Chain

A value-chain describes the steps that turn inputs into value-added output. For the software value-chain, we take the model of Berlecon Research (2002) as our point of departure.



The steps in this value-chain are:

- Software development: analysis, design, programming and testing of the software.
- Documentation: writing documentation (API documentation, Reference Manual, User Guides, Tutorials, Howto's, FAQs, ...)
- Software packaging: creating a user-friendly package of the software; bundling the software with other packages.
- Marketing/sales: marketing the software, closing sales, promoting wide-spread adoption, distribution.
- Consulting: providing consultancy with respect to the software.
- Integration/custom development: Integrating the software in the client's systems, customizing it for user-specific needs
- Training: training in the use or customization of the software
- Support: end-user support (telephone, e-mail), installation and update support, bug fixing
- Application management: operational management of the client's applications based on the software.

In the description of the business models we will indicate which steps of the value-chain are the focus in the business model. This focus will be on the production/programming or the services steps. We will not look into (actual or potential) business models that focus on the marketing and sales steps. Although such business models may be devised, we believe that such business models do not really depend on the specific features of F/OSS development and distribution models.

4.2.3 The Distribution Model

The distribution model describes how the company intends to reach its customers.

5 F/OSS Business Models

In the next sections we describe idealised “pure form” Business Models. Most companies will mix several of these pure form business models to create a sustainable business model. Several companies also mix these models with traditional closed source business models.

5.1 Business Model Overview

The table below gives an overview of the models that we have observed in the market and/or have been identified in the literature.

Model name	References	Alternative names	description	Prototype companies
Dual Licensing	Olsen 2006, Daffara 2007, Koenig 2004	Twin licensing	Companies make their F/OSS available under a non-reciprocal license	Sleepy Cat, MySQL
Support Sellers	Hecker 1999, Daffara 2007, Koenig 2004	Product specialists, Software provider (GPL)	Companies that provide paid support for F/OSS products they developed.	MySQL, Alfresco, Talend, Compiere
Third-party Support Seller	Krishnamurthy 2005	Third-party service provider	Companies that provide paid support for products they do not themselves develop.	EnterpriseDB, SpikeSource
Platform providers	Daffara 2007, Krishnamurthy 2005	Distributor	Companies bundle several F/OSS products into a complete platform. The company guarantees the quality of the integrated platform.	Red Hat, Novell, SpikeSource, SourceLabs
Consulting	Daffara 2007		Companies that provide consultancy w.r.t. F/OSS products based on their knowledge or experience of working with F/OSS	
Software as a Service	Koenig 2004	Hosted strategy	F/OSS is used to provide access to revenue-generating online services	Google, Salesforce.com
Added-Value providers	Krishnamurthy 2005	Software provider (non-GPL)	Companies that create proprietary software derived from F/OSS.	Apple, EnterpriseDB
Accessorizing	Hecker 1999		Selling services or physical items related to F/OSS (books, hardware, ...).	O'Reilly, Manning, SourceForge, CollabWeb
Loss Leader	Hecker 1999, Daffara 2007	Split OSS/commercial products	no-charge F/OSS product is used as a loss leader for traditional commercial software	Xen Source, SugarCRM, JasperSoft
Widget Frosting	Koenig 2004, Hecker 1999	Optimization Strategy	Selling a combined offering of high-value soft- and/or hardware components together with low-cost F/OSS offerings. The low-cost F/OSS enable higher pricing for the high-value components	Oracle, IBM

5.2 The Dual-licensing Model

In the dual-licensing model, the software product is available under two different licenses:

- a reciprocal open source license that obligates customers to release their own products also under the reciprocal license if they include the product as part of their own software products.
- a commercial license that releases the user from his obligation to release under a reciprocal license.

In short: either the customer reciprocates by contributing to the software commons or he pays the developers.

5.2.1 The Value Proposition

The customers gain the benefit of being able to incorporate F/OSS in their own offering without needing to open source these offering.

5.2.2 Product/services

The software product in binary and/or source code

5.2.3 Architecture of Value Creation

5.2.3.1 Market Design

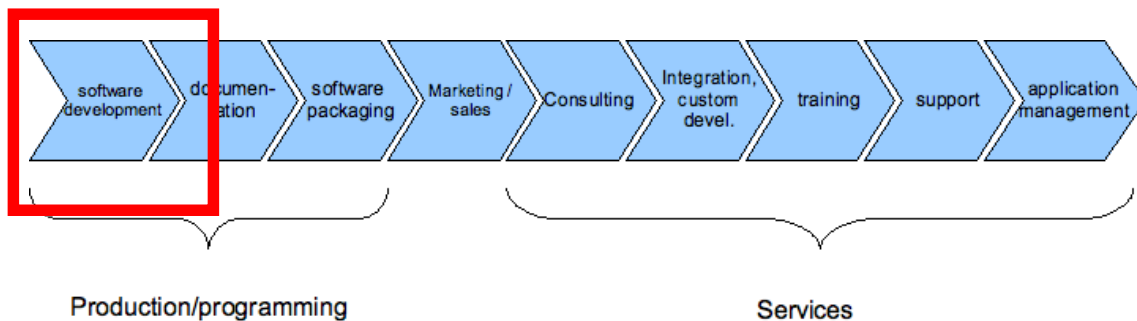
Geographical segmentation: global

Type segmentation: B2B

Software product segmentation: not relevant

5.2.3.2 Software value chain

A company that uses this business model needs to focus especially on the software development step of the value chain. The value of the product for prospective customers will increase with product quality and capability.



It should be noted that companies that use this model should pay particular attention to ways in which their product can function as a part in a larger software product or system.

5.2.3.3 Distribution model

The F/OSS distribution model (§2.5) makes the product widely known in developer communities, and helps to turn the product into a recognizable brand within the software professional community.

5.2.4 The Revenue Model

The revenue model is based on traditional licensing fees for the commercial licenses. The revenue model is the same model as used in equivalent closed-source companies.

5.2.5 The role of F/OSS licensing in the model

This model essentially uses the F/OSS distribution model to make the product a credible candidate as an input technology for the customer's own (proprietary) products.

If it is to work, it needs a reciprocal license (under an academic license such as the BSD license, the prospective customer cannot be forced to select the commercial license). Lastly, the model requires that the company holds the copyright to the software. This means in practice that the software cannot itself be based on third-party F/OSS source code under a reciprocal license.

5.2.6 Examples of companies that adopt the model

MySQL AB which provides the popular MySQL database. Sleepycat Software which provides the embedded Berkeley DB database. Sleepycat has recently been acquired by Oracle.

5.2.7 Strengths/weaknesses of the model

Strengths: provides licensing revenue with a possibility of high returns.

Weaknesses: requires complete ownership of the source code.

5.3 The Support Seller Model

In this model the company that creates a F/OSS product offers support services to users of the product. The model is based on the premiss that the creators of a software are the best suited to provide support because they are the creators.

5.3.1 The Value Proposition

An obstacle to F/OSS adoption for some classes of users is the lack of formal support. This is especially so for corporate and public sector customers. These customers usually prefer to source technology from a company that is accountable for defects and bugs. In this model the company ensures that F/OSS users receive the same level of support as is provided by commercial closed source companies.

5.3.2 Product/services

- Usually a standardised support service package for a F/OSS product. The package can include e-mail and phone support; automated support through a web system (e.g. automated notification of security updates); remote updating services. It typically includes a Service Level Agreement that specifies maximum initial response times to defect reports. A good example of such a service offering is the MySQL offering available at: <https://shop.mysql.com/enterprise/>
- Training course offerings

5.3.3 Architecture of Value Creation

5.3.3.1 Market Design

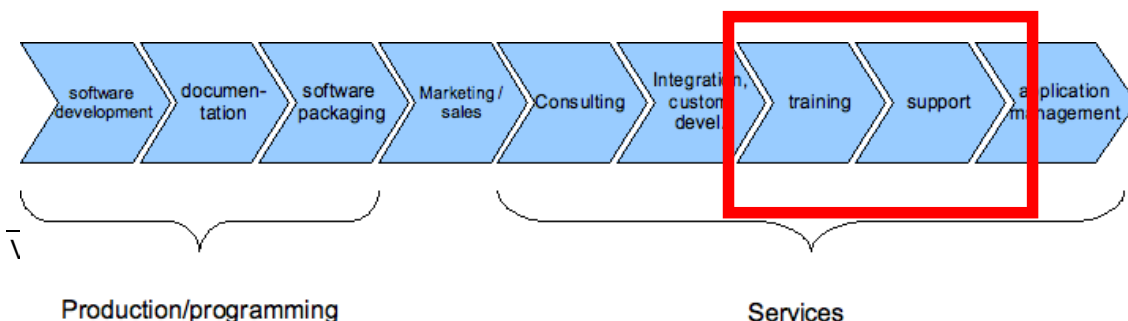
Geographical segmentation: global

Type segmentation: B2B and B2G.

Software product segmentation: enterprise solutions

5.3.3.2 Software value chain

Companies that adopt this model focus on the training and support steps of the value chain.



The software development does not in itself generates revenue since the product is offered purely as F/OSS. Nevertheless, it is very important because it establishes the credibility of the claim that the service offering is the best available.

5.3.3.3 *Distribution model*

The F/OSS distribution model is used to create a large installed base. This installed base can later be tapped for service support revenue.

This business model requires substantial marketing to convince users of the value of paying for commercial support.

5.3.4 The Revenue Model

Standardized support packages are offered as an SLA or support subscription for a fixed price on a (typically) yearly basis. This last model is the most important

5.3.5 The role of F/OSS licensing in the model

This model works with both reciprocal and academic licenses. Ownership of the complete code base is not necessary.

F/OSS is important in this model in two ways. As a development model, F/OSS allows the company to create a product quickly and with relatively limited investment. As a distribution model, it helps the company to establish a large user base. A fraction of that user base will be in the market for paid support services. A striking example of this is JBoss.com (now part of Red Hat) that creates the JBoss application server. After three years since operation, JBoss had achieved approximately 30% market share. After six years, it achieved revenues in excess of \$60 million, mainly from support contracts.

5.3.6 Examples of companies that adopt the model

MySQL AB (database systems), Alfresco (Content Management Systems), Talend (Extract-Transform-Load tools), Compière (Enterprise Resource Planning), JBoss (Java Application Servers; now a division of Red Hat). These companies created F/OSS products and generate revenue by selling support services.

JotSpot provides training in the Dojo library and many other Web 2.0 technologies that it helped create.

5.3.7 Strengths/weaknesses of the model

Strengths:

- revenue model allows high-margins since costs do no increase in proportion to revenue (a standardized package is offered at fixed-price).
- recurrent income since most users will renew the support contract.
- ownership of the code is less important; only effective technical leadership and know-how.

Weaknesses:

- requires substantial commercial effort to convince customers of the value of the offering.

- Value proposition only attractive for customers that use the product in business-critical systems and need reliable, high-quality support.
- there is a tension with the F/OSS spirit. The F/OSS community expects the developers to provide a measure of support and to fix bugs. If the non-paying F/OSS community members feel that their problems take a back seat to paying customers they will lose interest and faith in the product. On the other hand, if the community is large and active, the quality of community-provided support may obviate most user's need for professional support. This has been cited as one of the reasons why there is no Support Seller for the popular Postgresql database.
- there a low barriers of entry: the same offering can also be made by third-party companies (see the Third-party service provider model). In case of competition, the only competitive advantage of the company is its brand ("support by the creators of ..") and control of the technology.

5.4 The Third-party Support Seller Model

This is essentially the same as the previous business model with the exception that the company offering the support services is not the owner (copyright holder) of the supported F/OSS. The company need not even be part of the core development group.

This model is often used when the F/OSS product is either not owned by any one developer or development company, or when the owner of the source code is not able to provide professional support services (e.g. when he is an individual rather than a corporation).

5.4.1 The role of F/OSS licensing in the model

The free availability of the source code makes this model feasible. However, F/OSS dramatically lowers the barrier-of-entry for companies that want to offer competing F/OSS services.

F/OSS creates also the pre-conditions for this model. The F/OSS community creates the software and the installed base. The lack of formal, professional support then creates the market demand for these support services.

5.4.2 Examples of companies that adopt the model

Spikesource (<http://www.spikesource.com>) provides support services for a variety of open source packages for content management, customer relationship management, collaboration. These services are targeted at corporate or public sector customers. One of their offerings is targeted at the successful Drupal Content Management System. Drupal has been developed by several individual software developers, with a large number of source contributions from the community. No one can claim ownership of the Drupal source code, and the development is not controlled by a commercial entity. Spikesource provides for Drupal the professional support services that large corporations demand.

5.4.3 Strengths/weaknesses of the model

The strengths and weaknesses of the model are the same as for the support seller model, with the exception that the barriers to entry are even lower. The service offering can easily be copied by a competitor. When that happens, the company cannot differentiate itself as being in a privileged position w.r.t. the product, as is the case in the support seller model. This is an incentive for the company to get actively and visibly involved in the software development.

5.5 The Platform Provider Model

The company bundles several F/OSS products into a complete solution or platform. The company provides quality-assurances that the selected products work together. Companies such as Red Hat and Novell that provide commercial Linux distro's use this model.

This model is usually combined with the (Third-Party) Support Seller Model. First, because it is far easier to support and bug-fix a complete solution (platform) as it implies greater control over the operating environment. Secondly, the value proposition is enhanced for the customer if he can source the platform and related support services for the same supplier.

5.5.1 The Value Proposition

The value for the customer is that specialists have preselected and tested F/OSS components that together make up the solution or platform he is interested in. The company certifies that its bundle of F/OSS components work well together. In the absence of the product, customers would spend a lot of time and effort to have the various F/OSS components work together smoothly. And there would be no guarantee of quality.

5.5.2 Product/services

Pre-configured, tested and optimized bundles of F/OSS products that collectively provide a platform or solution.

5.5.3 Architecture of Value Creation

5.5.3.1 Market Design

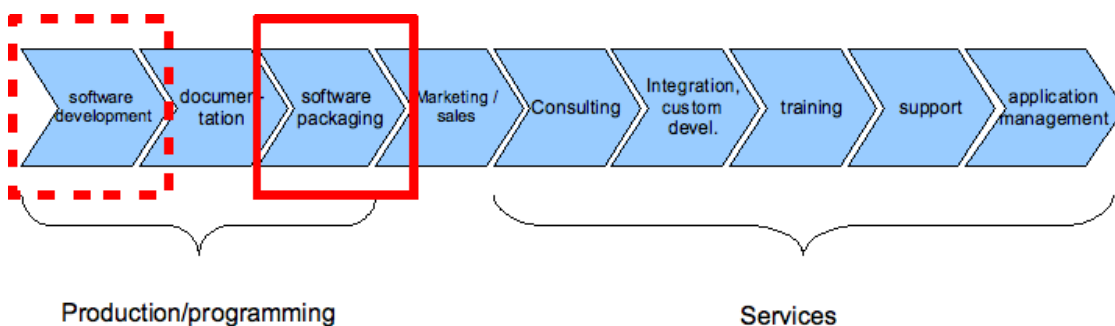
Geographical segmentation: global

Type segmentation: B2B, B2G

Software product segmentation: Enterprise Solutions

5.5.3.2 Software value chain

In this model the focus lies in the software packaging. Software development is also important. To ensure that the several components work together. This requires testing, bug-fixing and creating improved or optimized versions.



5.5.3.3 *Distribution model*

The model pre-supposes that the F/OSS products are already used in that combination and that a market for it exists. Significant marketing is required, however, to inform about the availability and advantages of the bundled solution.

5.5.4 The Revenue Model

Usually a license fee. However, the business model is mostly combined with a support seller model. In that case, the license fee will cover access to support services together with the bundled product.

5.5.5 The role of F/OSS licensing in the model

F/OSS creates the demand that this model address: an abundance of software where the problem for the user is not access to the software, but figuring out what offerings are of value, and what will work together seamlessly.

This model works both for reciprocal or academic licenses. However, in case of F/OSS constituents under a reciprocal license, the company must publish any improvements to these constituents.

5.5.6 Examples of companies that adopt the model

Red Hat and Novell provide Linux distributions geared for Enterprise usage (Red Hat Enterprise Linux, SUSE Linux Enterprise).

SourceLabs provides SASH: a complete optimized software stack for Java-based enterprise applications. It is combined of the following F/OSS products: Spring, Axis, Struts, and Hibernate. SourceLabs fixed a number of bugs, and made many improvements. An important point is that SourceLabs ensures that the most dependable versions of these products are combined in their product. Their slogan is: "Stop Wasting Time Self-supporting, Integrating & Testing".

Spikesource combines this model with the Third-Party Support Seller model. For example, their Drupal offering is named: Drupal Spikelgnited. From the product datasheet: "the Drupal Spikelgnited solution features SpikeSource tested, integrated, and certified open-source infrastructure components. SpikeSource evaluates each version of these components, integrates and configures combinations to work well with Drupal, and tests them ... to make sure the configuration meets your production needs." The Drupal offering includes also Drupal dependencies such as the MySQL database and LDAP.

5.5.7 Strengths/weaknesses of the model

Strengths:

- provides licensing revenue

Weaknesses:

- Added value may be too small to be attractive to customers. This is another reason, why it is usually combined with the Support Seller Model.
- Requires a large effort to make the quality assurance creditable in the market.

5.6 The Consulting Model

The company provides consulting and customisation services with respect to a range of F/OSS products. This model is certainly the most widely adopted model.

5.6.1 The Value Proposition

The customers is provided access to F/OSS expertise and software development skills.

5.6.2 Product/services

Consulting, customisation and bespoke software development services.

5.6.3 Architecture of Value Creation

5.6.3.1 Market Design

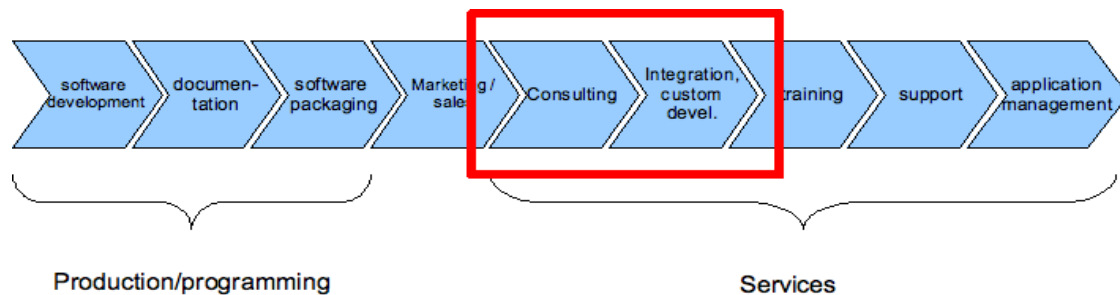
Geographical segmentation: local, depends on geographic proximity and a common linguistic/cultural background.

Type segmentation: B2B, B2G

Software product segmentation: Enterprise Solutions

5.6.3.2 Software value chain

Companies focus on the consulting, and integration/custom development steps of the value chain.



5.6.3.3 Distribution model

The distribution model is the same as for any other consulting company: customer relationship management, advertising, bidding on invited or open tenders.

5.6.4 The Revenue Model

Services are usually sold on a time & means basis. Custom developments are often contracted on a fixed price basis.

5.6.5 The role of F/OSS licensing in the model

The role of F/OSS in the Consulting model is that it lowers the barrier of entry. Consulting firms do not need to invest in expensive software products or developer tools. The availability of source code and (usually) a large body of published information on the software internals also mean that it is far easier to gain an expert-level understanding of the product.

5.6.6 Examples of companies that adopt the model

Accenture, Red Hat, Unisys and IBM are some of the big-brand companies that offer consulting services with respect to F/OSS. The model is also very widespread among SME's that target local markets. Most of these SME's are focused on Linux and other "infrastructure" F/OSS products.

5.6.7 Strengths/weaknesses of the model

Strengths:

- The model requires very low start-up costs and investments.

Weaknesses:

- The revenue stands in direct proportion to labour input, meaning that margins remain moderate.

5.7 The Software-as-a-Service Model

In this model F/OSS is used to create a web-accessible application service. Such systems are labeled "Software as a Service" (SaaS).

5.7.1 The Value Proposition

Sophisticated application functionality is provided on a low-cost basis. The customer also benefits from the fact that SaaS requires no IT administration or server hardware resources.

5.7.2 Product/services

Access to software as a service (SaaS).

5.7.3 Architecture of Value Creation

5.7.3.1 Market Design

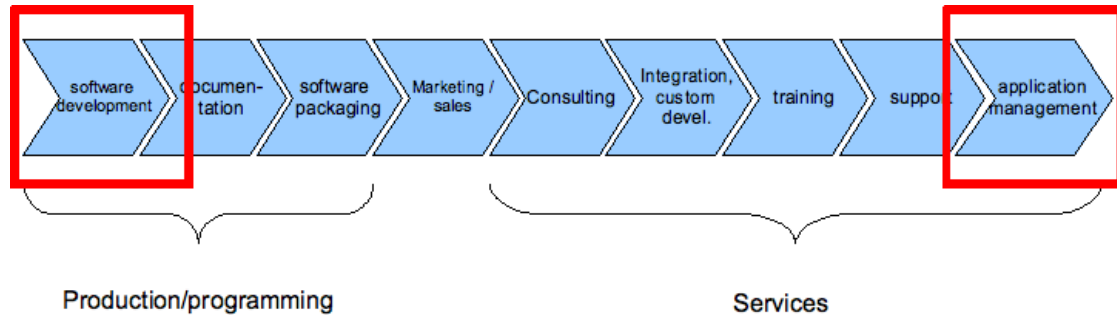
Geographical segmentation: global

Type segmentation: mainly B2B and B2G. Most consumer-oriented SaaS (such as flickr and YouTube) are free-of-charge to consumers and used to generate advertisement revenue (so are B2B from this point of view).

Software product segmentation: Enterprise Solution

5.7.3.2 Software value chain

For SaaS, the company need to develop the software service. The software development investment is significant because security, reliability and scalability are of paramount importance for the credibility of the offering. For these reasons, the the company also needs to invest in the application management step: reliable server infrastructure, networking, 24/7 application management,...



5.7.3.3 Distribution model

For this model to be successful the company must be able to build a strong brand. The company must convince the customer that he can entrust his data to it.

5.7.4 The Revenue Model

Usually the customer pays a monthly fee for access to the application services.

5.7.5 The role of F/OSS licensing in the model

F/OSS is very often used as a basis for creating SaaS services. Most reciprocal licenses, incl. the GPL (up to and including v3) impose no obligation that the software that implements the SaaS must be released as open source. Offering SaaS is not regarded as software distribution (this is called the SaaS-loophole).

5.7.6 Examples of companies that adopt the model

The best known company that uses this model is undoubtedly Google. Other examples are Salesforce.com for customer relationship management and Smallthought for database management.

5.7.7 Strengths/weaknesses of the model

Strengths:

- strong revenue model
- value proposition is easy to explain
- the “SaaS loophole” means that the company does not need to share the software for the service.

Weaknesses:

- Requires a strong brand to be successful
- Requires significant upfront investment in application development and infrastructure engineering.

5.8 *The Added-Value Provider Model*

In this model the company derives a proprietary software product that is derived from F/OSS.

5.8.1 The Value Proposition

The application functionality in so far as it is not present in the F/OSS from which the product is derived.

5.8.2 Product/services

Software product binaries and end-user license.

5.8.3 Architecture of Value Creation

5.8.3.1 *Market Design*

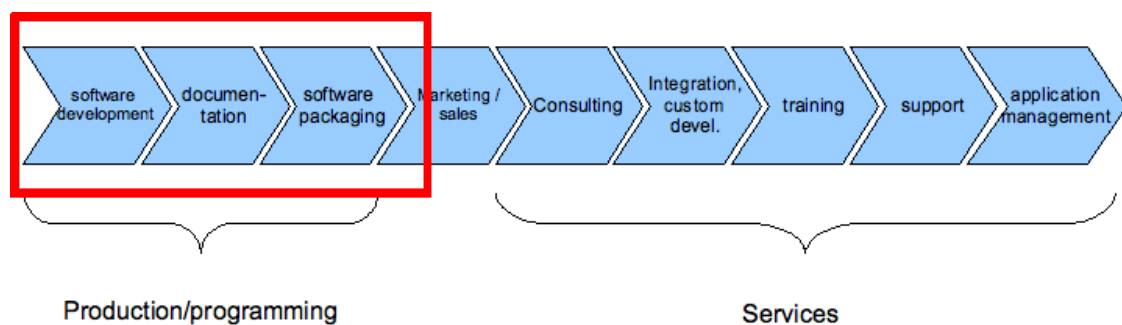
Geographical segmentation: global

Type segmentation: B2B, B2G, B2C

Software product segmentation: not relevant

5.8.3.2 *Software value chain*

The software company is positioned in the value chain like traditional software producers.



5.8.3.3 *Distribution model*

The company has the same distribution model as traditional software producers.

5.8.4 The Revenue Model

The revenue model is the same as for traditional software producers: licensing revenue.

5.8.5 The role of F/OSS licensing in the model

This model turns on the cost-reduction by using pre-existing F/OSS. In order to be able to appropriate the complete product the F/OSS must use an academic BSD-style license.

5.8.6 Examples of companies that adopt the model

Apple uses this model for its OS X operating system family. Apple used the FreeBSD as the core for OS X adding proprietary features such as the Quartz 2D graphics rendering technology, the Carbon application environment and the Aqua graphical user interface. Apple chose FreeBSD because it came with the BSD academic license that did not require Apple to open source these technologies, and allowed Apple to charge for the complete package.

5.8.7 Strengths/weaknesses of the model

Strengths:

- well-understood and proven business model
- licensing revenue model, allowing high margins

Weaknesses:

- Works only with academic licenses
- Same as for closed-source license models
- Requires significant branding to convince users to acquire the software
- Added-value must be significant because otherwise F/OSS projects will offer the same functionality for free.

5.9 *The Accessorizing Model*

The company sells physical accessories to F/OSS products. Most important of these are technical books and manuals.

5.9.1 The Value Proposition

In the central case of a technical publisher, the value is the access to first-rate technical information about the F/OSS product. The often chaotic nature of F/OSS documentation makes these books of value to users, esp. if the book is written by one of the core developers.

5.9.2 Product/services

Manuals, Technical literature.

5.9.3 Architecture of Value Creation

5.9.3.1 *Market Design*

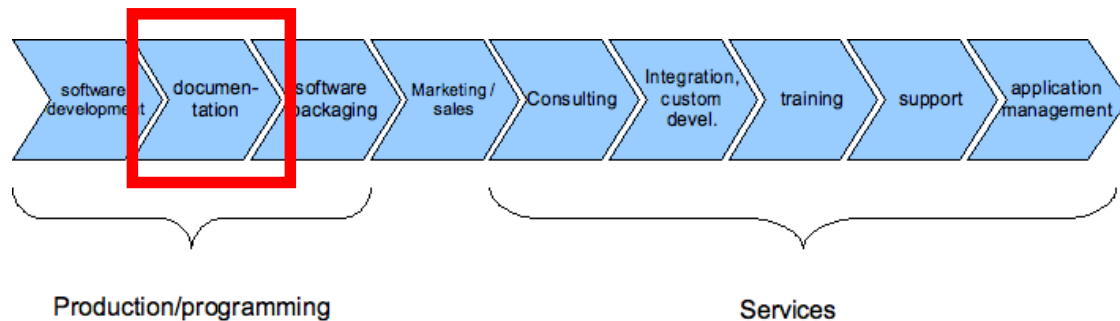
Geographical segmentation: global

Type segmentation: mainly B2C

Software product segmentation: not relevant

5.9.3.2 Software value chain

The company focuses exclusively on the documentation step of the value chain.



5.9.3.3 Distribution model

Same as for any other book publisher. When the book is written by one of the core developers, the publisher has the advantage that the book will be considered as especially authoritative.

5.9.4 The Revenue Model

Revenue from book sales.

5.9.5 The role of F/OSS licensing in the model

F/OSS licensing does not play a role. F/OSS creates the pre-conditions for this business model, not only by creating the subject matter for technical books but also, in some cases because of the absence of consistent high-quality documentation.

5.9.6 Examples of companies that adopt the model

O'Reilly and Manning are two publishers that have always paid particular attention to F/OSS-related books.

5.9.7 Strengths/weaknesses of the model

Strengths:

- Traditional, well-understood publishing model

Weaknesses:

- In some F/OSS projects the pace of change is so high that the books become quickly outdated.

5.10 The Loss Leader Model

In this model, a software company provides a F/OSS product as loss leader and market positioner for a proprietary product. The proprietary product offers additional high-value

functionality. The idea is that customers who need these functions are prepared to pay for it. The loss leader model is also used often in closed proprietary software business models.

5.10.1 The Value Proposition

The value proposition is the same as with traditional software companies. The model only changes the distribution model, not the value proposition.

5.10.2 Product/services

F/OSS for the loss leader, software product binaries and end-user licenses for the follow-up products.

5.10.3 Architecture of Value Creation

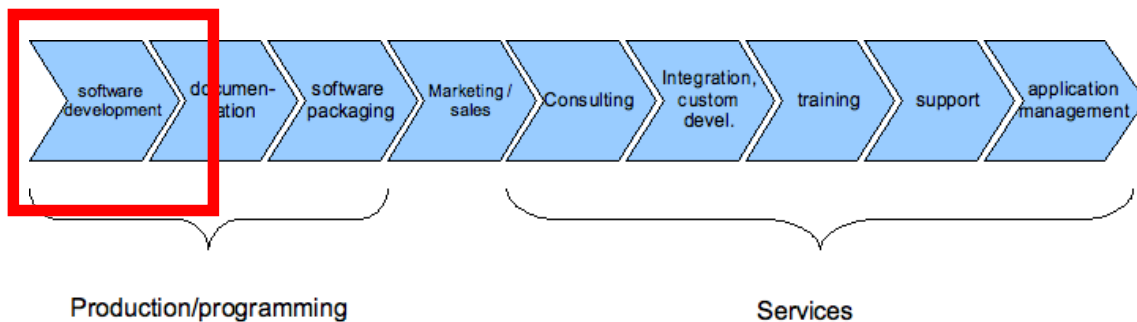
5.10.3.1 Market Design

Geographical segmentation: global

Type segmentation: B2B, B2G, B2C

Software product segmentation: not relevant

5.10.3.2 Software value chain



5.10.3.3 Distribution model

The loss leader creates market demand for the proprietary products. The F/OSS nature of the loss leader helps widespread distribution and helps building a brand for the company and the proprietary products within the software professional community.

5.10.4 The Revenue Model

Licensing (and support) revenues from the proprietary software packages that follow the loss leader.

5.10.5 The role of F/OSS licensing in the model

F/OSS is not essential for the loss leader strategy to capture market share. In fact most software companies that use the loss leader model keep the loss leader proprietary (e.g. Acrobat Reader, QuickTime Player, ESRI ArcExplorer, ...). Why do others then open source it? Here are some reasons:

- Open sourcing the product means that OSS Platform Providers, such as Linux Red Hat or Canonical (Ubuntu) can include the loss leader in their distribution. This ensures wide distribution of the loss leader.
- When the loss leader shares no code with the value-added proprietary follow-up products, F/OSS can lower the cost of developing the loss leader
- In a number of cases, the loss leader is a platform product offers generic and/or low-value application services. The company generates revenue from proprietary extensions or plugins to this platform. The F/OSS nature of the platform provides advantages in terms of initial development (by including other F/OSS products), and in terms of wide distribution.

The last reason show a common application of the loss leader that we will call the platform loss leader model.

5.10.6 Examples of companies that adopt the model

VMWare open sourced their base virtualization products in 2006 and used these as loss leader for their enterprise products. Their motivation to use open source loss leaders, is to have it included in popular Linux distributions.

Several companies provide proprietary, commercial plugins for the successful F/OSS IDE Eclipse.

5.10.7 Strengths/weaknesses of the model

Strengths:

- Well-understood revenue model (license revenue from proprietary software).

Weaknesses:

- Open sourcing a loss leader usually provides little incentive for the community to become actively involved in its development. Open sourcing only provides an advantage in terms of distribution and/or lower development costs.
- in the platform loss leader model, it is difficult to find a good balance between what to put into the F/OSS platform, and what to keep proprietary. The community around the platform product will want to put all functionality in the F/OSS platform, the company will want to keep certain functionality proprietary. In this context, managing community relations becomes very important. (In the case of Eclipse plugins, most companies that created proprietary plugins ended up open sourcing them for this reason). The platform loss leader model works best when the proprietary extensions are of very high-value, not easy to redevelop in the community, or encumbered by patents (e.g. Microsoft Exchange plugins for F/OSS groupware products).

5.11 The Widget Frosting Model

In this model a company sells a complete solution to a customer that typically includes both hard- and software. The solution contains F/OSS products as a means of reducing the price of the solution without lowering the company's profits. This model is adopted mainly by proprietary software vendors (SAP, Oracle) and System Integrators (EDS, IBM, Accenture, ...).

5.11.1 The Value Proposition

A complete hard- and software solution at lower cost.

5.11.2 Product/services

Complete enterprise solutions.

5.11.3 Architecture of Value Creation

5.11.3.1 Market Design

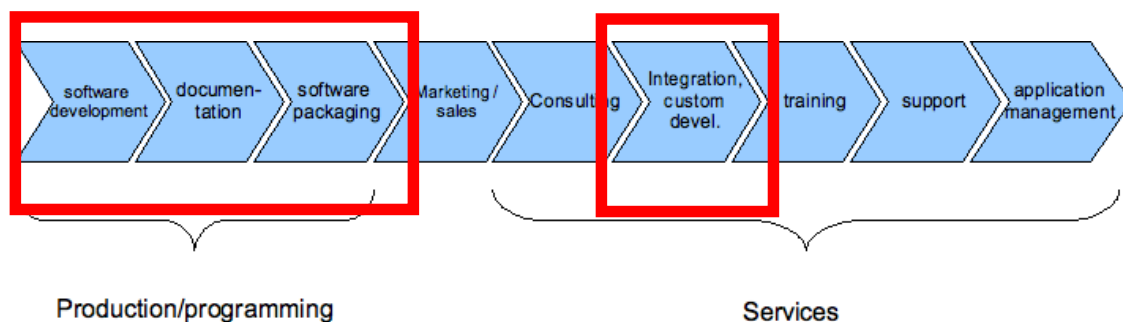
Geographical segmentation: global

Type segmentation: B2B, B2G

Software product segmentation: Enterprise solutions

5.11.3.2 Software value chain

The widget frosting model is a variant of traditional models of proprietary software vendors and System Integrators. The primary products that is sold is a proprietary software or hardware offering, or a custom-built solution.



5.11.3.3 Distribution model

The distribution model here is the model for system integrators: customer relationship management, advertising, bidding on invited or open tenders.

5.11.4 The Revenue Model

The revenue model is based on fixed price for the complete solution

5.11.5 The role of F/OSS licensing in the model

F/OSS provides high-quality, low-cost software components that reduce the cost of the solution.

5.11.6 Examples of companies that adopt the model

When Oracle was asked to deliver a database solution including hardware, it based its solution on the Linux OS. This enabled the company to strip out the Operating System as a cost in its offer.

5.11.7 Strengths/weaknesses of the model

Strengths:

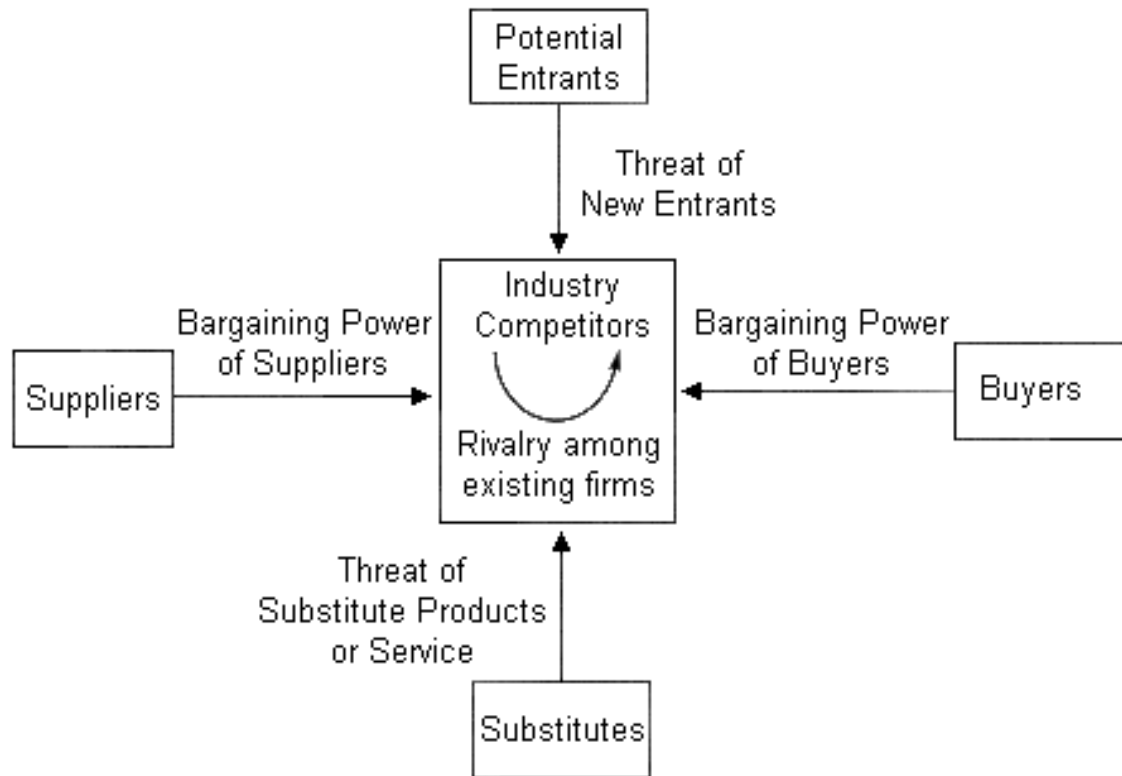
- Only part of the cost-reduction needs to be passed to the customer. The profit margins can be higher than alternatives with proprietary software instead of F/OSS.

Weaknesses:

- This model is only available for System Integrators and hard- or software vendors.
- Customer of the solution must be able and willing to work with the F/OSS components. That means in practice that the customer must be offered professional support options with respect to the F/OSS components, or the company itself must provide support for the complete solution, incl. the F/OSS components.

6 The Effects of F/OSS on the software market

The Five Forces model of Porter is a model that describes the structure of a market outside-in.



The model distinguishes the following forces:

- Threat of new Entries: how easy or difficult is it for competitors to enter the market?
- Bargaining Power of Suppliers: How strong is the position of suppliers?
- Bargaining Power of Buyers: How strong is the position of buyers?
- Rivalry among Existing Firms: How strong is the competition between firms in the market.
- Threat of substitutes: how easy can the product of service be substituted?

Using this model, we describe in the next sections the effect of the emergence of F/OSS on the traditional software product and services markets. We distinguish between product and service markets because the service provider is usually not the same as the software product vendor.

6.1 Threat of new Entries

F/OSS dramatically lowers the barriers to entry in the software product market, because a F/OSS software creator can incorporate pre-existing F/OSS into his product offering. Red Hat is an example. Red Hat can create an enterprise ready Operating System with an estimated investment of \$60 million a year because the F/OSS ecosystem provides an estimated \$1 billion development.

In the software service market, F/OSS has the same effect. The software is freely available and its source code can be examined. The open nature of F/OSS development also means that no privileged vendor-information exists that blocks new entrants from offering services.

6.2 *Bargaining Power of Suppliers*

F/OSS reduces the power of suppliers in software product and service markets. F/OSS provides a large commons of freely available software that limits the power of established software vendors. If the pricing of supplier software vendors becomes too high, market actors will either revert to F/OSS alternatives, or

6.3 *Bargaining Power of buyers*

Mirroring its effect on suppliers, F/OSS increases the bargaining powers of buyers. An illustration is provided by several large public sector bodies that used the threat of moving to Linux to exert pressure on Microsoft to lower its cost.

In the software service market, the openness of the F/OSS development model makes information freely available with regard to the capabilities, complexity and limitations of the software. This information can be used by buyers to assess more accurately the value of a service offering.

6.4 *Rivalry among Existing Firms*

In the software service markets F/OSS promotes rivalry because it is so easy to enter the market and start competing. Proprietary software vendors can exert a measure of control on who offers product-related services by establishing Value-added Reseller partnerships and a distributor network. ESRI, for example, allows just one official distributor within a geographic area, and grants these distributors monopoly rights for providing support services within that area. In the case of F/OSS products, such vendor control over the service offering is virtually impossible.

In the software product market, the availability of F/OSS reduces the monopoly power of high-profile proprietary software companies such as Microsoft.

6.5 *Threat of substitutes*

The emergence of F/OSS made the creation possible of substitutes for some products and services. These substitutes are usually in the form of SaaS. F/OSS is important for the emergence of these substitutes because it provides the necessary scalability at low cost.

7 Open Source Business Risks

7.1 License compliance

Open Source is not public domain software: it comes with a license that imposes obligations on whoever uses the software. A company that wants to build a business on open-source should verify whether its approach is consistent with the licenses of all open source software it uses. Here is an example: company X offers bespoke software development services for local government. Normally, bespoke software development for a client who uses the software internally only is permitted by the GPL because there is no distribution of the software. But because the software could be used by many of its customers, company X uses a different model. It pre-finances the software development and charges its customers licensing fees (the customers are contractually obliged to buy the software). In other words, it distributes its software, and that means that in order to do this legally within the bounds of the GPL, this company must also make its source code available. That would obviously destroy the whole model. If this company wants to take advantage of open source, it must either change its model, or avoid all open source software covered by the GPL (or other reciprocal licenses).

In cases where a product is created on the basis of a mixture of open source software packages, each with their own license, the problem of license compliance becomes even more difficult. If you have GPL, MPL, and Apache licenses, under which licenses can you distribute your own product? Can you legally distribute such a product? The Free Software Foundation claims that Apache 2.0 license and the GPL 2.0 license are incompatible so they may not be combined in a distributed product (The Apache Foundation disagrees with the FSF's analysis).

7.2 Liability

Most open source licenses contain a “No liability claim” stating the the software is provided “as is”, and that the creators are not liable or any damages caused by the program. In Europe, at least, companies that use open source for their services and products are not protected by such clauses. First because European consumer-protection law overrides such “no liability” claims, and secondly because a commercial service or product based on open source product is distinct from the open source product, and therefore may imply in itself new liabilities. So what happens when you create an product or service based on open source package X, and bugs or malfunctions in X causes damages to your client? Can you disclaim liability? Probably not. You incur liabilities when using open source in a commercial offering, and these liabilities cannot be referred to the creators of the open source software.

7.3 Patents

When the business model involves the creation of open source software, the biggest legal risk is the risk for patent litigation. The big software companies own so many patents that it is virtually impossible to be absolutely certain that the software doesn't infringe on any patents. The cost of defending against patent litigation is so high that start-ups and smaller F/OSS companies are not able to bear this cost. To mitigate the effect of software patents, the F/OSS community has included patent provisions in F/OSS licenses. For example, the GPL version 3 contains provisions to the effect that licensors must provide all users with any patent licenses necessary to run, modify or distribute the software. In addition, a patent holder is required not to bring patent suits against other users on the GPL on pain of having all his GPL licenses automatically revoked. It is uncertain how helpful these provisions will turn out to be. Perceived wisdom is that users of F/OSS have little to fear from patent licenses. Creators of F/OSS, however, should conduct some investigation into patents that may impinge on the project. Considering the number, complexity and sometimes vagueness of software patents, such a research cannot hope to be conclusive, however. It is only a best-effort attempt to identify relevant patents. Any identified patents that seem to restrict the software can then be designed around. This approach is probably the best that can be hoped for in the present circumstances.

8 Open Source Business Models in Practice

Currently, the most important open source models in terms of creating business value are the Support Seller and the Platform Provider Models. Both models hold the promise of achieving high profit margins because they offer a standardized product at a fixed price. These models also enable to convert technical leadership of a F/OSS project into a revenue-generating offering. For these reasons, the leading companies in the Open Source market are based on these models and their variants. Among these are MySQL AB, Alfresco, Mulesource, Red Hat, Canonical, and Pentaho.

The support seller and platform provider are natural business models in the software market because they provide a necessary compliment to the community focus of F/OSS projects. The community-driven nature is great for the technical development of a project: lots of users, contributors and developers co-operate to create good software that is responsive to the user's real needs. *Corporate* users of F/OSS, however, find it uncomfortable to have to deal with a community to gain access to software support services. They would rather establish company-to-company relations that provide accountability, stability and dependability w.r.t. support. Support Sellers and Platform Providers are the commercial entities that corporate users can build such relations with. It is also for this reason that companies such as Spring21, Alfresco and Mulesource count many Global 2000 companies among their clients: such large, often multinational firms understand the value of F/OSS software but require contractually-guaranteed support, sourced from dependable firms. (This has a flip-side: it's largely the larger companies that constitute the market for support sellers. SME's, being small, agile and flexible, find it not so hard to do without formal support contracts, and therefore are not easily persuaded to buy support contracts).

Interestingly, the value of these Support Sellers and Platform Providers does not seem to reside primarily in their intellectual property but in their brand. This became clear when the Oracle Linux offering was announced last year. Oracle pledged to create a new Linux distro that was the same as the Red Hat distribution but at lower cost. Unfortunately for Oracle, their offering has only had very limited success in the market place. The reason seems to be that even at the lower price, the offering isn't very creditable. No one creates a better Red Hat Server than Red Hat. Customers value Red Hat and its products because Red Hat has proven its ability to provide a well integrated, tested and fully-supported enterprise-ready operating system - that is what the Red Hat brand stands for. So far, Oracle has been unable to convince the market that they can offer the same value and reliability as Red Hat. The value of the brand has important implications for Open source companies. First, technical leadership of the community around a project remains very important because that helps building the brand and underpin its credibility to customers. The second lesson is that the Third-Party Support Seller model may not be commercially feasible if a creditable Support Seller that maintains leadership over the project is already established.

Although the most interesting open source business model are the Support Seller or Platform Provider models, the most widely adopted is the Consulting Model. This is illustrated by the large number of consultancy firms that sell Linux-related consultancy services. The openness of the F/OSS enables many small entrepreneurs to start offering services without negotiating ISV or partner status and with very limited investments. The model is available, so to speak, to every one with a laptop and an internet connection. These consultancies create a sort of ecosystem around the F/OSS project. F/OSS-based consultancies offer services that sit between the Support Seller and the community-provided service. They can provide professional support services that are closely tailored to the local market and clients, but will not achieve the high margins or the global reach of the Support Sellers.

9 Application to the geospatial market

The geospatial market is a niche within the software market. It is characterized by high product complexity and a user base that is heavily concentrated in the public sector and the scientific community. On the other hand, there is also a lot of market power in the hands of a small group of software companies with ESRI being the clear market leader.

If we look at F/OSS companies in this market, we find no clear winners - companies that are comparable to a JBoss or Red Hat. The companies that we most readily associate with geospatial open source, such as Refrations, Vivid Solutions and Lat/lon are SME's and have adopted a business model that is essentially the Consulting Model (support and development services on an ad-hoc basis). This model puts a limit to the growth that they can realise, and it also means that they cannot capitalize on the popularity of their products (such as PostGIS or the JTS) in areas remote from their home base (Europe, Asia).

What could account for the absence of Platform providers and (third-party) Support Sellers in the geospatial market? Surely, a market dominated by the public sector – often large customers that value the dependability and stability of contractual relations – makes this an ideal market for the emergence of support sellers. So why has no such company emerged?

The first reason is that the vendor lock-in of GIS is such that (especially) the large GIS users find it very hard to switch over to F/OSS GIS. This reason cannot be the whole story because we must now ask: why is it so hard to switch to open-source? Could it be that the GIS F/OSS community has failed to provide a software suit that is comparable to the commercial software suits? I believe that this is the case w.r.t. desktop GIS, high-quality cartography and data-editing tools. Tools like GRASS and QGIS provide rich spatial analysis and easy to use GIS viewing, resp., but they don't come close to the full richness of ArcGIS as a spatial analysis, cartographic map production and data-editing tool. In other functional domains (programming API's, libraries, spatial databases and web mapping) the F/OSS offerings are comparable or even superior to the closed-source offerings. Many customers, unfortunately need desktop GIS, data-editing and high-quality cartography software. They will go to commercial GIS vendors for this type of software, and GIS vendors will then try to exploit their position to maximally influence the rest of the GIS infrastructure of the customer. (The reason that ArcGIS Editor needs a Geodatabase has probably more to do with ESRI's desire to impose a proprietary format on the database than with any technical requirements). So as long as no creditable F/OSS alternative exists for desktop GIS, cartography and data editing, no end-to-end F/OSS based software stack for GIS can be offered. This, in my view, effectively precludes a Red Hat-like company in GIS.

Acronyms and Abbreviations

BI	Business Intelligence
BSD	Berkeley Software Distribution
CPL	Common Public License
CRM	Customer Relationship Management
ERP	Enterprise Resource Planning
F/OSS	Free/Open Source Software
FSF	Free Software Foundation
GPL	GNU General Public License
IDE	Integrated Development Environment
LGPL	Library (or Lesser) GNU General Public License
MPL	Mozilla Public License
OSI	Open Source Initiative
OSD	The Open Source Definition
SaaS	Software as a Service
SLA	Service Level Agreement
SME	Small to Medium Enterprise
UML	Universal Modelling Language

References

- Baldwin, C. Y. and Clark K. B. Does Code Architecture Mitigate Free-Riding in the Open Source Development Model? Harvard Business School Working Paper, available at: <http://www.people.hbs.edu/cbaldwin/DR2/BaldwinClark.ArchOS.Jun03.pdf>, 2003.
- Behlendorf B. Open Source as a Business Strategy. In DiBona, C., Ockman, S. and Stone M. (eds.), *Open Sources: Voices from the Open Source Revolution*, pp. 113-126. O'Reilly & Associates, Sebastopol, 1999.
- Berlecon Research Gmbh. Basics of Open Source Software Markets and Business Models – FLOSS Final Report, available at: <http://www.infonomics.nl/FLOSS/report>, 2002.
- Bessen, j. Free Provision of Complex Public Goods. In Open Source Software: Do Firms Practise What they Preach? In Bitzer, J. and Schröder Ph. (eds.): *The Economics of Open Source Software Development*, pp. 83-109. Elsevier, Amsterdam, 2006.
- Daffara D. Business models in FLOS-based companies. Working paper, available at: <http://opensource.mit.edu/papers/OSSEMP07-daffara.pdf>, 2007
- Fogel K. *Producing Open Source Software*. O'Reilly & Associates, Sebastopol, 2006.
- Gosh, R. A. and Prakash, V. V. The orbiteen free software survey. *First Monday*, 5(7), 2000.
- Gosh, R. A., Glott, R., Kreiger B. and Robles, G. The free/libre and open source software developers survey and study – FLOSS Final Report, available at: <http://www.infonomics.nl/FLOSS/report>, 2002.
- Healy, K. and Schussman, A. The Ecology of Open Source Software Development. Working Paper, available at: <http://opensource.mit.edu/papers/healyschussman.pdf>, 2003.
- Hecker, F. Setting up shop: The business of open-source software. *IEEE Software*, 16(1): 45-55, 1999.
- Hunt, F. and Johnson, P. On the Pareto distribution of sourceforge projects. In *Proceedings of the Open Source Software Development Workshop*, pp. 122-129, Newcastle, UK, 2002.
- Koenig, J. Seven open source business strategies for competitive advantage. Available at: <http://www.itmanagersjournal.com/feature/314>, 2004.
- Krishnamurthy, S. An Analysis of Open Source Business Models. In Feller, F., Fitzgerald, B., Hissam, S. A. and Lakhani K. R. (eds.): *Perspectives on Free and Open Source Software*, pp.279-296. MIT Press, Cambridge USA, 2005.
- Lakhani, R., Wolf, R.G. Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects. In Feller, F., Fitzgerald, B., Hissam, S. A. and Lakhani K. R. (eds.): *Perspectives on Free and Open Source Software*, pp.279-296. MIT Press, Cambridge USA, 2005.
- Lerner, J., Tirole, J. Some simple economics of open source. *Journal of Industrial Economics*, 50 (2): 197-234, 2002.
- Mockus, A., Fielding, R.T. & Herbsleb, J. A case study of open source software development: the Apache server. In *Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000)*, pp. 263-272, Limerick, Ireland.
- Olsen M. Dual Licensing. In DiBona, C., Cooper D. and Stone M. (eds). *Open Sources 2.0: The Continuing Evolution*, pp. 71-90. O'Reilly & Associates, Sebastopol, 2006.
- Osterwalder, A. The Business Model Ontology. PhD Thesis, Université de Lausanne, 2004.
- Raymond, E. S. The Cathedral and the Bazaar. *First Monday*, 3(3), 1998.
- Rosen, L. *Open Source Licensing*. Prentice Hall, Upper Saddle River, 2004.

Rossi, C. Bonaccorsi, A. Contributing to the common pool resources in Open Source software. A comparison between individuals and firms. Working Paper, Sant'Anna School of Advanced Studies Institute for Informatics and Telematics (IIT-CNR), Pisa, Italy, 2004.

Rossi, C., Bonaccorsi, A. Intrinsic Motivations and Profit-Oriented Firms. In Open Source Software: Do Firms Practise What they Preach? In Bitzer, J. and Schröder Ph. (eds.): *The Economics of Open Source Software Development*, pp. 83-109. Elsevier, Amsterdam, 2006.

Scott, B. The Open Source Legal Landscape. Available at:
http://opensourcelaw.biz/publications/papers/BScott_OSS_Legal_Landscape_060328.pdf, 2006.

Stähler, Business Models as a Unit of Analysis for Strategizing. Paper presented at the International Workshop on Business Models, Lausanne, Switzerland, 2002.

Stallman, R. What is copyleft? Available at <http://www.gnu.org/copyleft/copyleft.html>, 1984.

Young, R. Giving it Away: How Red Hat software stumbled on across a new economic model and helped improve an industry. In DiBona, C., Ockman, S. and Stone M. (eds.), *Open Sources: Voices from the Open Source Revolution*, pp. 113-126. O'Reilly & Associates, Sebastopol, 1999.