A critical approach to Open Source software

Stefan Görling

Submitted for the degree of Masters of Science
June 2003

Abstract

The purpose of this thesis was to discuss a number of assumptions regarding the benefits of Open Source software projects. By studying what has been written about Open Source combined with a number of own data collections, this thesis argues that:

- Brooks law is still valid in Open Source projects
- Many Open Source projects are failures
- Open Source culture is a product of the 90s, not the 70s
- Open Source is no guarantee for reduced lock-in effects
- Our most famous Open Source projects are not built up by nerds working for free, but professionals, employed by commercial companies to contribute to the projects.
- Large Open Source projects are often hierarchical and bureaucratic
- Opening your source does not automatically lead to a large number of contributors
- Open Source breeds diversity, not a single winner
- Open Source projects often targets the community itself, rather than external actors
- Companies benefiting from Open Source are often based on traditional business models rather than revolutionary visions
- Open Source is not necessarily an efficient way to develop software

Acknowledgments

Here I would acknowledge my cat if I had one.

Contents

List of Figures

List of Tables

Chapter 1

Introduction to Open Source

> *Begin at the beginning and go on till you come to the end; then stop.*
>
> –Lewis Carroll

The purpose of this chapter is to give you, the reader, an introduction to what Open Source is and how it is used. This introduction is rather short, and it might therefore not be sufficient to explain everything you need to know about Open Source in order to enjoy this thesis to its full extent. If I have failed to provide the necessary pieces here, please take a look at the reference list.

1.1 What is Open Source

Open Source could be described as a collective effort to develop software in an efficient way. In order to maximize the benefit from the software, the developers chooses to give away the immaterial rights to their works to anyone, often provided that they do the same if they improve the software.

What Open Source is, depends on the perspective one uses. Various aspects of Open Source could be described as a methodology to develop software, a culture or even a religion.

The Open Source trademark is protected by the Open Source Initiative, who describes the basic idea behind Open Source as [32]:

> "The basic idea behind open source is very simple: When programmers can read, redistribute, and modify the source code for a piece of software, the software evolves. People improve it, people adapt it, people fix bugs. And this can happen at a speed that, if one is used to the slow pace of conventional software development, seems astonishing."

The perhaps most famous Open Source project is the Linux Kernel, and various projects surrounding it. The project were created by Linus Torvalds in 1991 and have now developed into a popular operating system, used by millions people. Linus posted the first piece of source code to the public later on that year and since that day, thousands of developers have joined the project by contributing with fixes and new features.

The Linux Kernel project have often been used as the primary example of the efficiency of the concept of Open Source, and the conclusions made from this project have often been generalized into all areas of software development by various writers on the subject.

The most common way to describe Open Source is to look at it as a combination of a methodology and a culture.

Open Source methodology is described as a highly decentralized development system, enabled by the Internet. By a common versioning control system, developers over the whole world can keep their local copy of the source code and develop on their workstation. When they have a feature that they want to add, or a bug they have a fix for, they send a request to the maintainer to include the new code into the source code tree. The maintainer acts as a relay for approving additions to the source code, in large projects there might be multiple levels of maintainers, each responsible for a certain part of the code.

The Open Source culture could be compared to the cultures of athletics. Everyone who have been a member of a teem sport can tell you that the actual game, is only a part of the reason why they train,

the social activities around the sport, communication with people sharing a common interest, is often as important as the actual game. Developing a project is not only an interaction with the computer, but with a society of other developers working on the project. The Open Source cultures have a low level of tolerance for free-riders, and strong ethic opinions about sportsmanship and fair play. There is a high level of technical reasoning, striving to always include the best technical solution.

1.2 A example of Open Source in corporations

To give a better description about how Open Source could be beneficial for corporations, I will illustrate an example of a small software development company here. The type of company is chosen as it is easy to show how they may benefit from using Open Source in their products.

We will begin the journey with the picture of a traditional company (figure 1.1).

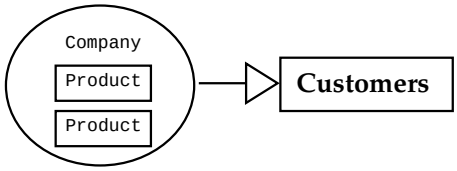


Figure 1.1: A traditional software development company

Our traditional company is very simple, it is a software company developing various software solutions for their customers. The product is built from scratch in-house and the company owns the full rights to the products.

When our simple company faces a problem that is too large for them to handle themselves. Say that they need to incorporate a database engine in their products, in order to store and process data in an efficient way[1]. As a database engine is complex, the company chose to license a third-party product rather than inventing the wheel themselves (figure 1.2).

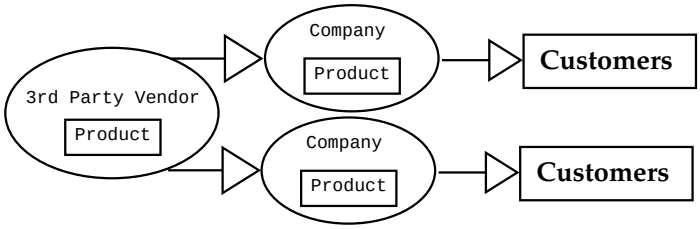


Figure 1.2: A software development company using third party components

As the database engine vendor specializes in delivering solutions to other companies, they are able to split up the cost on multiple customer, thus reducing the cost for every single party, as well as generating a, often rather large, profit for the database vendor. The drawback for our company is that they have no, or little, power over what the third-party vendor chooses to do in the future.

The other choice available for our company is to use a Open Source product (figure 1.3).

If the company could find a Open Source project which to some extent solves their problem, they could join the project and use that product as a database engine for their project. If the project does not solve all the needs of the company, they will have to add the features themselves, or hire someone to do it. But as long as those additions are less expensive than buying the whole product from a commercial vendor, it might be beneficial to choose a Open Source product. The main benefit is that you are in complete control of the product, you could modify it as you wish. The main

[1] A database engine such as Oracle, is a good example as the product generally is considered to complex, and requires specialization. Even huge companies such as the business system vendor SAP do not make their own database engines, but utilizes third party solutions.

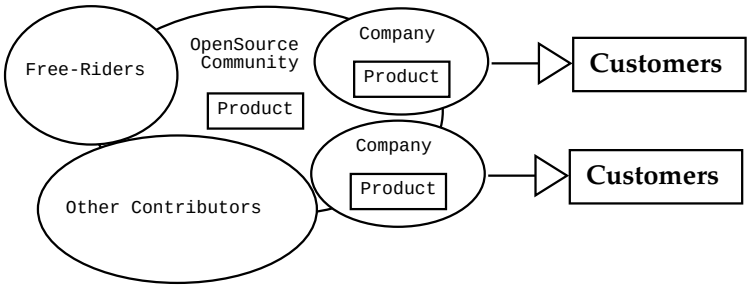


Figure 1.3: A model of an Open Source community model

disadvantage is that if there is nobody else is interested in the same features, you most likely will have to add them yourselves.

1.3 A short note on licenses

An introduction to Open Source could not be complete unless we discuss the foundation that keeps it all together. How do we ensure that free-riders do not steal the work done by the community and leaves off.

The intellectual property created by programmers is always protected by copyright, an individual can not resign their copyright rights to their works. Instead of trying to resign from their rights, they have used the copyright system to their benefit. By releasing their works under various licenses, stating exactly what one may do with the source code, they can rest assured that nobody could violate their rules.

There are a large number of licenses used, the GNU Public License and the Lesser GNU Public License are the most popular ones[2].

Those licenses are constructed to ensure that the software is and remains available to the community, the various licenses differ to some degree but the Open Source Initiative have defined a number of basic principle a license should follow in order to be called open [33].

1. Free redistribution
2. Access to source code
3. Must allow derivate works
4. Must protect the integrity of the author
5. Must not discriminate certain users
6. No Discrimination against fields of endeavor
7. Distribution of license
8. The license must not be specific to a product
9. The license must not restrict other software
10. The license must be technology-neutral

It is important to understand that the license is a very central issue when it comes to Open Source development. At a quick glance one might interpret the license as restrictions on how you are allowed to use and distribute the provided source code. To a lawyer, this contract is a simple legal document, regulating what you may and may not do with the software. But to many developers, it is

[2]See Appendix B

much more than that. The decision on what license to use is a highly ideological decision, different licenses reflects different views on what is right and wrong in the world of Open Source.

Software licenses are often divided into two main categories, viral and non-viral licenses.

Viral licenses is the most restricted license. The name originates from the fact that once a piece of software is "infected" with a small snippet of source code licensed under a viral license, the whole project must be relicensed under this license and given back to the community. This restriction disables many commercial companies from using this kind of license, as it forces them to give away their whole product if they borrow a single module containing a viral license.

The non-viral license is a slightly less restricted, it requires that improvements in the current code is to be given back to the community, but that other modules using the code could be licensed under another license.

1.4 When to choose Open Source

After this short introduction, the question remaining unanswered is the single one pure economists really are interested in: when shall we go for the open way, when shall we go for the third-party way and when should we do everything in-house.

Lucky for us, pure economics are built up by rather simple, logic rules, and the answer is as obvious as the question: when the profit of choosing the open way, is greater that the cost. This of course simply ignores that the actual estimate of costs and gains are the hard part, but that is business economics as usual.

This chapter will not go deep below the surface on when to choose Open Source, and when to not do it, as my main goal is to question whether the widely used assumptions are true or not. Starting of by using them to discuss the benefits and disadvantages of Open Source might not be all that bright.

Open source have been used in many companies, for many years, without anyone caring about it, as this kinds of decisions were often delegated down to the IT/IS staff. As IT/IS have become a critical success factors for many companies, managers on many levels have started to care about the strategies and policies, making this decisions managerial rather than technical.

Chapter 2

Method and Delimitations

> *The researches of many commentators have already thrown much darkness on this subject, and it is probable that if they continue we shall soon know nothing at all about it.*
>
> – Mark Twain

The purpose of this chapter is to define the methods and delimitations I will use for this thesis.

2.1 Background

The increasing interest in Open Source software is full of success stories and strategies instructing companies in how to get their free lunch by converting their strategies in order to benefit from the Open Source software (OSS) community. A few years ago, talking about Open Source were a foolproof way to raise the value of your stocks. When the hype of the new economy crashed, one might have believed that it would be the end of the Open Source hype too. Instead, the hype have increased, so that even governments around the globe are evaluating it in order to scare Microsoft to lower their prices. Open Source is often described as a revolution, i.e. something which radically changes all our previous knowledge.

2.2 Research Question

Does Open Source methodology and business models fundamentally differs from traditional software development, is it necessarily better?

2.3 Description

The aim is to study the conceptions of OSS and determine whether they are misconceptions or not. OSS is often described as a very effective methodology to create superior products, however, initial research shows that there are much redundancy and bureaucracy in this environment.

Much have been written regarding Open Source software, its people, its ideologies and methodology. As this field of research still is emerging the last years have been something of a golden era for scholars and journalists exploring this area, often blinded by the hype. Many scholars of today are building their works on this foundation built during the era of the New Economy, affected by the early IPO success of companies which today either are dead or at least struggling to survive. Is the praise about Open Source well grounded?

2.4 Phases

1. Identify the mythical properties of the free lunch, what is said to be significant about Open Source Software. How is the phenomena covered in press and literature.

2. Question the properties from a critical perspective both by discussing the rhetoric and by collecting data from the Open Source communities (OSC) themselves.

3. Exemplify how the difference between myth and reality can affect decisions regarding opening the source code by discussing the release of the Netscape Browser.

2.5 Delimitations

The properties of Open Source could be discussed to an infinite length, in order to be able to narrow it down I will single out a limited number of assumptions to focus on.

2.6 Method

The main emphasis of this thesis is to discuss how current writers on the topic reasons on the topic, therefore a major part of the study will be carried out by studying what has been written on the subject so far. In order to discuss the culture, two quantitative investigations will be carried out.

2.6.1 Investigation 1: Who is paying for free software

The aim of this study is to identify to what extent Open Source software development is carried out by amateurs of professionals. By studying the activity on the various development mailing-lists for a number of the most popular Open Source projects[1], I will be able to identify the most active persons and their employers, thus creating a picture of how many of them that are working for free.

2.6.2 Investigation 2: A study on Open Source projects that never took off

The other study will aggregate statistics from over 35.000 Open Source projects hosted on Sourceforge.Net, the largest community for Open Source development projects. By studying those projects I will be able to provide statistics about not only what kind of software projects and platforms that are most popular, but also how long projects usually live before they either become a success or a failure, as well as how many developers there usually are in a Open Source project.

2.7 Potential conclusions to be reached

As the era of the new economy have exaggerated the concept of Open Source, I expect to find out that some of the properties of the culture and methodology isn't as beneficial for commercial companies as earlier described. My initial research shows that the vast majority of Open Source projects dies long before they become viable software solutions. My research further determines that most of the developers working on the most popular Open Source projects are actually paid by various software companies, the Open Source culture have been converted into a loosely connected system of co-operation between companies, rather than hobby projects for computer nerds.

I also expect to find out that the effects of Open Source have been largely exaggerated, and that some of the assumptions actually are based on myths, while some proves more viable.

[1] Such as the Apache Web server, Linux Kernel, SAMBA file server, GNOME desktop, WINE windows emulator. Full data is included in appendix A

Chapter 3

Open Source praise

> *The Linux philosophy is to laugh in the face of danger. Oops. Wrong one. "Do it yourself." That's it*
>
> – Linus Torvalds

The purpose of this chapter is to discuss, identify and exemplify what statements I have found to be significant assumptions when discussing the concepts of Open Source.

Recently there has been an increasing buzz in the media about Open Source, companies are investing large amounts of money and resources into an area which have yet not been fully explored. Governments all over the world is investigating how to replace their existing software in order to lower their costs. There have been a number of studies on Open Source from various aspects, but some of them have been written rather hasty and sometimes lacking a homogeneous perspective. In this chapter I will present some praise about Open Source in general to illustrate that it is in fact a praised topic, as well as focus in on a few, what I call myths, topics in special which I will return to in the following chapters.

IBM is one of the companies which openly states that they are investing huge amounts of money on Open Source software. In year 2000, IBM announced that they were to spend over 1 billion USD on Linux related activities during the next year, and that all their servers should be able to run Linux.

> Open source is a **revolution in software development** - a revolution well underway. In fact, many of the Internet's key technologies were developed, and continue to evolve using open source methods
>
> – IBM [15]

Even though Microsoft initially met the success of Linux and Open Source first with ignorance, and then with FUD-marketing, they have recently turned around and now acknowledges this new movement as a serious competitor to their de facto monopoly. They are now trying to find new ways to compete with something that is free and cannot be acquired, which is a fundamentally new situation for them.

> "Linux is a serious competitor," said Ballmer. "We have to compete with free software, on value, but in a smart way. We cannot price at zero, so we need to justify our posture and pricing. Linux isn't going to go away–our job is to provide a better product in the marketplace."
>
> He acknowledged there was more to Linux than free software–the main benefit of the open-source movement was the community developing software and sharing ideas. "Linux is not about free software, it is about community," he said. "It's not like Novell, it isn't going to run out of money–it started off bankrupt, in a way."
>
> – Steve Ballmer, Microsoft [16]

Hewlett Packard have also been keeping their eyes of Open Source software, they are not often seen in the Open Source communities in the same way as IBM, nor do they feature Linux servers in their TV-commercial, but they are committed to Open Source as an idea that changes the current game in the software industry.

> Open Source is a revolutionary perspective on how software should be created. It defies the stereotypes of what is to be a technical person in the twenty-first century. It challenges the very foundation of the current software industry.
>
> Using Linux and open source technologies, thousands of enterprises are cutting costs, gaining flexibility, and discovering powerful new sources of business value.
>
> – Martin Fink, Hewlett Packard [10]

To give you a taste of what the media is writing, I have added a quote from News.com below. There are many others, too many to mention them here, but the content is often resembling what Olson has to say.

> Quietly, over the past two years, open-source software has made an enormous difference in the way businesses operate. It's already deep in the Internet. More importantly, though, smart companies are finding ways to use open source directly. They're building better products faster–and more cheaply–than they ever could before. If you're not at least thinking about doing that yourself, you could be in trouble.
>
> – Michael A. Olson, News.com [31]

The first potential myth I will look into is that Open Source is often market as a bullet-proof way to get a pool of developers, contributing to your project, giving back state of the art computer code which you could migrate back into your commercial products. The community is described as a huge pool of developers, spending their time on contributing on different computer projects, without any economical motives at all. If i only open up my software, everybody will start using it and contribute to it.

- **Myth:** If I give away my software to the Open Source community, thousands of developers will suddenly start working for me for nothing

The Open Source methodology has been described as the most efficient way to develop large software projects in an efficient manner. Using the Linux kernel as an example, writers often concludes that the Open Source way of doing things is the most efficient way to develop software. Much of this comes from the essay "The Catherdral and the Bazaar", by Eric S. Raymond [36], one of the chief ideologists originating from the culture itself. Raymond concludes that Brooks law [3], which stated that adding developers to a software project only adds complexity and that development therefore must be carried out by single individuals or very small teams, have finally been over-proved.

As Linux is marketed as a better operating system than those provided by Microsoft, one presses the point that Open Source development projects will be able to develop, and if left alone, will produce state of the art software. For example, investigations have shown that the networking code in Linux have fewer errors in the code than many commercial operating systems [37].

Raymond further reasons in his introduction why he decided to sit down and write his book:

> What I saw around me was a community which had evolved the most effective software-development method ever *and didn't know it!* [36]

Hewlett Packard have accepted this fact and are trying to create an internal Open Source community, using Open Source methods in-house as a way to increase creativity and efficiency [23].

- **Myth:** Open Source methods is a highly efficient way to develop software

The beauty of Open Source is that everyone can contribute, and that the best solution will be selected. This makes the software evolve in a Darwinistic way, where the strongest pieces of source code will be accepted into the program, hence making the program the best possible solution.

Russel C Pavlicek writes that:

> It does not matter whether the best solution comes from someone with three Ph D's and 30 years of experience or from a bright 18-year-old who is just beginning college. The project will benefit most by using the best solution. The pedigree of the author makes little difference. The issue is the solution [34, p. 15]

Which greatly exemplifies the theory that the best (technical) solution will always win and make it into the project, no matter from whom it may come. This has been regarded as one of the greatest benefits with Open Source. The survival of the fittest.

- **Myth:** The best solution will always win

One successful part of proprietary software and hardware vendors is that they lock you into a specific platform. When you buy a Playstation game console, it will only be able to play games which have been certified by Sony, the producer of the console. The benefit of locking in the end user is obvious, if the user chooses your platform, and it is locked, you ensures that the customer will return to you, no matter how satisfied they are with your services. As long as buying upgrades from your current vendor is cheaper than replacing the whole platform with other vendors, the customer will stay with you. The more Playstation games you buy, the less likely is it that you will buy another console instead, and by making the next generation backwards-compatible, you further lock in the customer into your platform. This is why game consoles are often sold at a price below cost of production, as it will lead to revenues from new games for years to come.

I will try to avoid comparison between Open Source and Microsoft, as it is like comparing Microsoft to lets say, Balanced Scorecard. One could compare companies basing their products on Open Source, with Microsoft, but you can not compare a company and a methodology or culture.

However, as Microsoft is the master example of this strategy, they will here be used as an example of lock-in effects. Microsoft with their Office Suite they hold a monopoly of office software such as word processors and spreadsheet software. The document format of the Microsoft Office programs is secret, and although it has been reverse engineered to a certain level, it is virtually impossible to ensure that another office suite is 100% compatible. And for every letter written, for every spreadsheet created, you are further locked in to the platform. As long as the cost of licensing Microsoft Office is less than the cost of converting all already produced documents as well as tutoring all users in a new system, the customers will stay. Hence, making their products more proprietary enables them to increase prices.

Many countries are adopting policies in order to make sure that Open Source products are evaluated together with proprietary products when considering purchases.

> Open Source Software within UK Government, published by the Cabinet Office this week, spells out key decisions over open source use by the Government, including its intention to "seek to avoid lock-in to proprietary IT products and services". [18]

- **Myth:** Open Source avoids lock-in effects

Another strength of the Bazaar model, or Open Source model, is said to be the concept of releasing early and often, and that this policy enables users to examine and test the software continually during the development process and thereby contribute with bug-fixes and ideas. By tightening the feedback loop, and enabling users to act as co-developers, your users becomes more committed, and they can help improve the software.

> The Bazaar model allows motivated users to examine and test the software while it is in development [34, p. 11]

This assumption was one of the reasons Netscape decided to open their browser. Brian Rehlendorf developer on the Apache project argues that:

> Furthermore, depending on the license applied, you may see lower costs involved with development of your software. You're likely to see bugs fixed by motivated customers, for example. You're also likely to see new innovations in your software by customers who contribute their code to the project because they want to see it maintained as a a standard part of the overall distribution. [6, p. 155]

- **Myth:** The Bazaar model allows motivated users to examine and test the software while it is in development

The story that a bunch of nerds, working for free, is threatening the Microsoft empire, is a catchy slogan which have been popular among journalists. It is easy to pitch stories such as the revenge of the nerds, or the cyberpunk communists. As a legacy from those stories, Open Source projects is often described as ruled by pimpled nerds working for free all nights long, without any monetary motives.

It all began when Linux where to be explained to the masses, and the poster boys actually were nerds working for free. TV series such as “Triumph of the nerds” and “Revolution OS” [25], best selling books like “Accidental Empires: How the Boys of Silicon Valley Make Their Millions, Battle Foreign Competition, and Still Can’t Get a Date”, or Torvalds own book , “Just for fun” [42], helps to nurture this myth.

In March 2003, Business Week had an issue titled “The Linux Uprising” carrying the subtitle

> How a ragtag band of software geeks is threatening Sun and Microsoft–and turning the computer world upside down

- **Myth:** The Open Source communities is built up by nerds working for free

The myths lives on.

Chapter 4

Results from data collection

Get your facts first, then you can distort them as you please.

– Mark Twains

The purpose of this chapter is to summarize collected data that I have gathered.

4.1 Open Source developers

One of the few studies that has been done on Open Source developers was carried out by the Boston Consulting Group (BCG) in cooperation with the Open Source Developers Network (OSDN) [1]. The study was done as a web questionnaire running on the popular websites run by OSDN. Even though the study have been questioned by some developers[1], it could be used for some basic conclusions regarding the developers demographics and motivations. A more detailed study were done in the EC founded FLOSS project, and may be found in Section 4 in their final report [18].

The age span of Open Source developers are rather large, they come in all ages. There is a large cluster of developers in the ages of 20-35, with strong male dominance. (est. over 95% of the active developers). Two fifth of them are singles, the rest, around 60 % lives in some kind of partnership, which to some extent defies the rumour of introvert nerds running the Open Source communties.

The ratio of paid vs. unpaid work various between different projects. The most famous and mature Open Source project as the Linux Kernel have a large amount of professional contributors[2], where smaller, hobby projects often have none. There seems to be a connection between the maturity/popularity of the project and the ratio of paid contributors, although this has not been thoroughly investigated.

Project	*% professionals*	*% amateurs*
The Apache Webserver	68	32
The GNOME Desktop	48	52
The Linux Kernel	86	14
The SAMBA Server	64	38
WINE	33	67

Table 4.1: The ratio of paid vs. unpaid developers in various Open Source projects.

Table 4.1 indicates that what once were pure hobby projects, have evolved, due to the impact of commercial interests, into a loosely connected system of cooperation between various companies interested in the technology offered by the project.

While there are a large share of professional developers in those few popular projects, other investigation shows that there are a larger share of hobbyists in other, smaller, projects.

Our most famous Open Source projects are not built up by nerds working for free, but professionals, employed by commercial companies to

[1] Claims were that some of the questions and alternatives were altered during the survey.

[2] See Appendix A for full information.

The BCG Study clearly shows that the Open Source community is not a homogeneous environment, they chose to group the respondents into four subgroups [1, p. 18]:

- Believers, stating that code should be open for ideological reasons.
- Skill enhancers, hacking to learn.
- Fun seekers, hacking for joy.
- Professionals, hacking for money.

This grouping is done as there is almost as many opinions on motivations and reasoning as there are respondents to the study. I believe that the main conclusions from this data is not that those four groups could be identified, but that the terms hacker and Open Source developer should be used with care, as the meaning could be very diverse. It is possible to divide people into different groups depending on how they reason regarding their development efforts, but the generalizations will be very inexact as many people tend to affiliate themselves with many of those groups.

The question of motivation is often being discussed, for example, a recent article on the community website Slashdot argues that there on a single day were two new articles discussing this matter [2], reaching mutually exclusive conclusions.

As the most famous essay on Open Source development, The Cathedral and the Bazaar [36], is based upon an authors own experience with a Open Source project, many other introspective articles have followed thereafter. Many developers states that the challenge is what really motivates them. Once they are presented to the problem, they feel that they could create a better solution themselves, rather than using something existing.

Raymond starts by suggesting that:

> Every good work of software starts by scratching a developers itch.

Soon to be acknowledged by famous hacker Larry Wall [6, p. 127]

> Three great virtues of programming are laziness, impatience and hubris.

Many of us recognize the inspiration we will get if someone tells us that we can not do a certain thing. It is something that have motivated people for centuries, doing stuff not only to impress others, but to show themselves that they can do the impossible. A programmer bright enough not to be challenged by school, could easily find themselves challenges in the world of Open Source software. When Richard Stallman, the authority on free software, spoke up that a task was to difficult, someone where bound to step up to the task.

> Stallman said that the job was to difficult - it would require a complete rewrite of all the tools, and it would be too difficult to maintain. Gumby told him it wasn't such a "Big F*cking Deal" and hence named this new creation the BFD library. (We explained to our customers that BFD stood for the binary file descriptor library.) [6, p. 81]

4.2 Open Source projects

The media often focus on a few established, successful projects. The reason is obvious, it is easier to identify them and it is easier to tell the story of a hero. Success is more interesting than failure. In order to gain a wider understanding on how many Open Source projects actually fits in to the patterns of the larger projects I have studied a larger number projects, of various sizes and success. The study contained roughly 34 000 projects hosted on the largest Open Source developer community on the net[3].

My first finding is that most projects are small, over half of the studied projects are single man projects, over 75% have less than three developers. Less than ten per cent have five or more developers.

Projects	*Percentage*	*Aggregated*	*Developers*
20242	59.08 %	59.08 %	1
6223	18.16 %	77.24 %	2
2797	8.16 %	85.41 %	3
1515	4.42 %	89.83 %	4
952	2.78 %	92.61 %	5
575	1.68 %	94.29 %	6
444	1.30 %	95.58 %	7
1244	3.63 %	99.21 %	Other

Table 4.2: A table showing how many developers the projects have. Projects with more than 7 developers have been grouped as Other.

Over time, only a third of the registered projects have had any activity. Most project are completely dead or have very low activity.

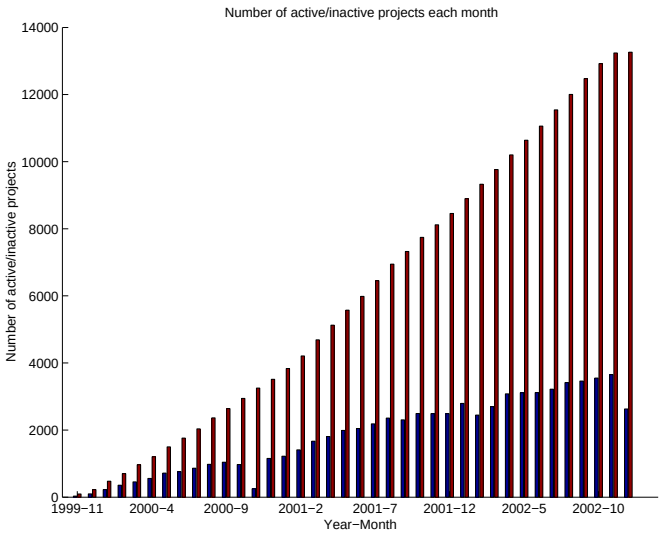


Figure 4.1: A figure illustrating the number of active vs. inactive projects. The high bars denotes inactive projects.

Many Open Source project are failures

Open Source projects fails for many reasons, if the project maintainer fails to create a community around the project it might die as soon as the maintainer have solved the problem, which he had in mind when he created the project. Projects also fades away since the developers switches to another project with similar interests.

[3] See Appendix B for the full report.

Table 4.3 shows that only a third of all Open Source projects targets the end users, most projects are tools for developers or system administrations. Furthermore, most of the projects is targeted towards users Unix operating systems such as Linux and BSD rather than, Microsoft Windows (table 4.4).

Audience	*Percentage*	*Aggregated*	*Projects*
20755	38.08 %	38.08 %	Developers
18044	33.11 %	71.19 %	End Users/Desktop
8833	16.21 %	87.40 %	System Administrators
4488	8.24 %	95.64 %	Other Audience
762	1.40 %	97.04 %	Information Technology
558	1.02 %	98.06 %	Education
1057	1.94 %	100.00 %	Other

Table 4.3: A table showing who the projects are targeting. Items below 1% has been grouped as Other

Targeting developers and system administrators is of course the easiest way to build a community around a project. If the users actually are developers, they are able to read the source code and might just help out with changes and fixes. After all, that is what they are paid for. What is interesting to note here is that most of the efforts are kept within the community itself. Most of the projects are targets the persons within the system, not the great masses of end-users outside the communities.

Open Source projects often targets the community itself, rather than external actors.

Operating System	*Percentage*	*Aggregated*	*Projects*
13288	27.94 %	27.94 %	Linux
12181	25.62 %	53.56 %	OS Independent
4471	9.40 %	62.96 %	Windows 95/98/2000
3624	7.62 %	70.59 %	POSIX
3405	7.16 %	77.75 %	Windows
2551	5.36 %	83.11 %	Windows NT/2000
1148	2.41 %	85.53 %	SunOS/Solaris
852	1.79 %	87.32 %	MacOS X
825	1.73 %	89.05 %	FreeBSD
676	1.42 %	90.47 %	BSD
664	1.40 %	91.87 %	Other OS
594	1.25 %	93.12 %	Microsoft
3272	6.88 %	100.00 %	Other

Table 4.4: A table showing which Operating Systems projects are targeting. Items below 1% has been grouped as Other

The fact that most efforts are targeted to end within the community, not mainly for people outside it, is a fact we must consider carefully when illustrating Open Source as altruistic communism, or as a gift-economy. Before assuming that people are intending to give away the fruits of their work to total strangers, we must assert whether this intention exists or not[4].

Projects do follow the principle of early releases, time before first download is often less than one month within the registration of the project. This is in full accordance with the principles of releasing early and releasing ofter in order to make enable others to see what you are doing and helping out if they share your visions.

Even though a large part of the projects have stalled, they have not reached mature state. Most of the stalled projects are in the planning or pre-alpha stage (figure 4.5). This is important to show that

[4]An sarcastic analogy: Even though we throw out our garbage and rats might benefit from this, we are not considering garbage disposal as a gift-economy.

projects not only die as they are finnished, but that progress ends long before the program becomes stable.

Month	*Percentage*	*Aggregated*	*Projects*
7259	34.62 %	34.62 %	1 - Planning
4735	22.58 %	57.20 %	2 - Pre-Alpha
3241	15.46 %	72.65 %	3 - Alpha
3057	14.58 %	87.23 %	4 - Beta
2365	11.28 %	98.51 %	5 - Production/Stable
269	1.28 %	99.79 %	6 - Mature
43	0.21 %	100.00 %	7 - Inactive

Table 4.5: A table showing the maturity of the projects which are inactive.

4.3 Open Source communities

Another main reason for the success of the Open Source culture as of today is the evolution of the Internet. Even though people sometime meets in person on conferences and occasional travel, most of the communication is carried out through the Internet. The Open Source culture could be described as a form of transactional community, which may initially sound like an oxymoron. Kollock argues that a community is built up by persons sharing the same values, and that Internet-based relations and communities may indeed be as real as traditional communities [19]. The Open Source community is built up around the transactions involved by coordinating the development, and the discussions on how the project should evolve. After a while, people might start saluting each other and asking some questions on social matters such as how their vacation were.

Let us go back to the definition of community. Pollock concludes that a community is built up by persons sharing the same values. There is really nothing in the definition of community itself that requires it to be built upon trust-relations, even though this have traditionally been the case.

The FLOSS report further shows that most of the developers response that they began their Open Source development during the 1990s, and most of them were at “university”-age when they began. The Open Source culture is said to be rooted in the academic environment, and it does share many values with the academic world in general. Another reason why the universities have been active recruiters might be that during university education, there is often free access to computer resources coupled with lots of free time and freedom to focus on hobby projects. The universities themselves might not be active recruiters, but the environment they provide for their students has been proved to be a solid base for Open Source engagements.

The rapid growth during the 90s is probably partly because of the birth of the Open Source label, before the 90s, Open Source hackers where merely hackers. Earlier, there were not enough hackers with similar interests, in the same district, so that they could form a community. Internet changed all that, communities can now be formed around more narrow topics than ever before, fans of flintknapping could join together and discuss their interests with peers[5].

Other reasons for the growth during this time is of course the success of Linux, as well as the heavily increased number of computer science students during the era of the new economy. As we have seen, most of the Open Source projects are not targeted toward end-user systems, but toward Unix/Linux systems, an area which were disabled for non-professionals before the birth of Linux and its competitors[6].

Many writers dates the concept of Open Source back to 1960-1970, due to the fact that the spirit of the early university computer labs praised sharing of resources. To be historically correct, we must pay some tribute to the early hackers as there are sometimes looked upon as ancient gods, but the culture as we know it today, is a product of the last 10-15 years.

[5] Actually, they have, on http://www.flintknapping.com

[6] There were Unix systems available before then, such as MINIX and various BSD variants, but they simply did not take off in the same way, so timing was of essence.

In fact, there are only a very minor number of people still referring to "the early days", primarily Richard Stallman, originator of the GNU initiative, followed by a few others. Even though Stallman often yells as if he were a large crowd, he is becomings becoming more and more isolated as people, even though they do recognize his efforts in the past, identifies themselves rather as a member of the culture of today, a world-wide distributed community, than a member of the culture that existed in a single lab in MIT before they were born.

There is something of a schism between the remaining oldies representing the old culture, and the majority of the developers who are younger. It could be described as idealists versus pragmatists[7]. Stallman tries to achieve his almost religious utopia, while Torvalds and many others simply wants to get things done in the best way possible. Torvalds comments upon Stallman in his book:

> The thing that drives me crazy about Richard is that he sees everything in black and white. And that creates unnecessary political divisions. He never understands the viewpoint of anybody else. If he were into religion, you would call him a religious fanatic. *[42, p. 195]*

Open Source culture is a product of the 90s, not the 70s

The fact that the culture is not as it were in the 1960s is significant as many works are based upon this assumption. The most famous description of the 1960s hacker culture is Levy's book "Hackers, heroes of the computer revolution" [21]. This book is based on the stories told by Richard Stallman and others in the MIT A.I. Lab, the homebrew computer club and other communities of that time. The work is cited in many pieces about Open Source, which should be an indicator for taking precautions.

Considering Open Source communities as a product of the 90s enables us to analyze it from the perspective of how communities of today are evolving, and how the youth of today looks upon community engagements. Putnam investigates how most communital engagement is declining, and how we are becomming increasingly reluctant to commit to this kind of engagement [35].

If community efforts are diminishing, we are becoming increasingly transactional and afraid to commit, how come Open Source survives where other communities fails? There are a number of properties of Open Source which matches the current state of society.

Keeping money out of the system, or at least not as one of the primary motivators, Open Source enables people from all phases of life to participate, without the need of large amounts of founds. This does not rule out that people are active Open Source developers even when they enter the working age, but what is somewhat unique is that money is never a central issue the same way as it is in sports where various participation fees and competition prizes are common, embedded elements of the game. As there are no initial financial investments, participants are free to enter and leave at a very low transaction cost, hence risk. As we are becoming more risk averse when we feel less secure in our environment, this is definitely attractive. Of course, as we have entered the community, and begin to build relations, our fear for the uncertainity motivates us to stay rather than searching for new community.

One common assumption is that when working life takes an increasingly large part of our life, we replace our traditional social relations through associations, by relations to our co-workers. Then Again, Putnam argues, with what I believe is justified, against the idea that our workplaces is becoming our new arena for new relations as there are social restrictions in a work-place. Therefore, as social animals, we need to find another arena for social activity, which fits our habits of life. I believe that Open Source communities are constructed in a way that meet this need.

The status system of Open Source communities is far more transactional than more traditional civic associations. We are no longer interested in investing many years of hard, unpaid work in order to gain social status sometime in the future unless we are outmaneuvered by someone else. Doing voluntary work without a guarantee of social status is a risk greater than we will accept. In Open Source communities one gain reputation for what you actually do, your transactions adjusts your status.

The concept of fun is fundamental when it comes to Open Source. People often tries to find pseudo-rational explanations why they are doing things, Open Source developers have converged

[7] I am sad to admit that this description was not invented by myself. Torvalds referes to it in his book, telling that it is not his either, but without mentioning the original source of the statement.

around the concept of fun as reason. The transactional community actually enables us to take part in a community, at the same time as we are not breaking the rules of play. It is carried out on freely accepted rules, and we are free to leave at any time we prefer, at the time we shut or computer down, or stops reading the newsgroup, we are back in reality. This is in full accordance with the rules of play as identified by Huizinga [13] and Callois [4].

4.4 Open Source business models

> If the collapse of the Berlin Wall had taught us anything, it was that socialism alone was not a sustainable economic model.
>
> – Robert Young, Red Hat

Making money on a large number of people who is working for their own joy has been proved to be very hard. The problem has a striking resemblance with the problems of making money on sports[8], where you have the same problem with non-paid workers. I will ignore all ludicrous business models which received seed capital during the era of the new economy, only to fade away quickly when the streams of money diminished. The companies who survived, could be divided up into a small number of business models. The models which have survived seems a lot less visionary and revolutionary than one might expect from this revolutionary way of conducting business.

Companies benefiting from Open Source are often based on traditional business models rather than revolutionary visions

4.4.1 Services for the players, the communities and the games

Every community needs a number of basic services to function well. Open Source developers needs hosting services, development tools, community websites in order to function as a group and be able to develop software efficient[9].

4.4.2 Repackaging and distribution

One of the easiest way to earn money from the communities is simply to repackage and distribute what the team is doing so that it is available for a wider public. As well in sports as in Open Source projects there is a large difference between the number of active participators and the potential number of people interested in it or benefiting from it. In sports we have small core of players, pursuing a game which a smaller audience have the possibility to enjoy, as they know that there is a game, and manages to get hold of a ticket. Open Source shares the same attributes (figure 4.2), there is a core development group, which contains the main players, a larger number of contributing users who use the software and occasionally fixes bugs, as well as a huge number of potential users, which could benefit from the software if it was only packaged the right way (figure 4.3).

The gap between the core and the potential users is a obvious place where there is money to be made. And therefore there are companies working with packaging and distribution of the software/games[10] as well as other user groups which works to narrow this gap down[11].

There are a number of reasons why people chooses to buy a product from a vendor rather than downloading it themselves. If we are to depend on a certain piece of our software for business critical applications we often prefer to have someone to call when there is a malfunction. Therefore there is a huge market for repackaging Open Source software. Or as Robert Young, CEO of Red Hat says [6, p. 114]:

> Drinking water can be had in most industrial countries simply by turning on the nearest tap, so why does Evian sell millions of dollar of French tap water into those markets? It boils down to a largely irrational fear that the water coming from your tap is not to be trusted.

[8] A more in-depth discussion about the relation between Open Source and play can be found in Appendix C

[9] Companies like Cygnus Solutions, Bit Keeper and the Open Source Development Networks (http://www.osdn.net) represents examples of this strategy.

[10] For instance Red Hat Software and Eurosport

[11] Soccer news at http://www.soccernews.com, Linux Install Fest at http://linux.ucla.edu/events/installfest.php3

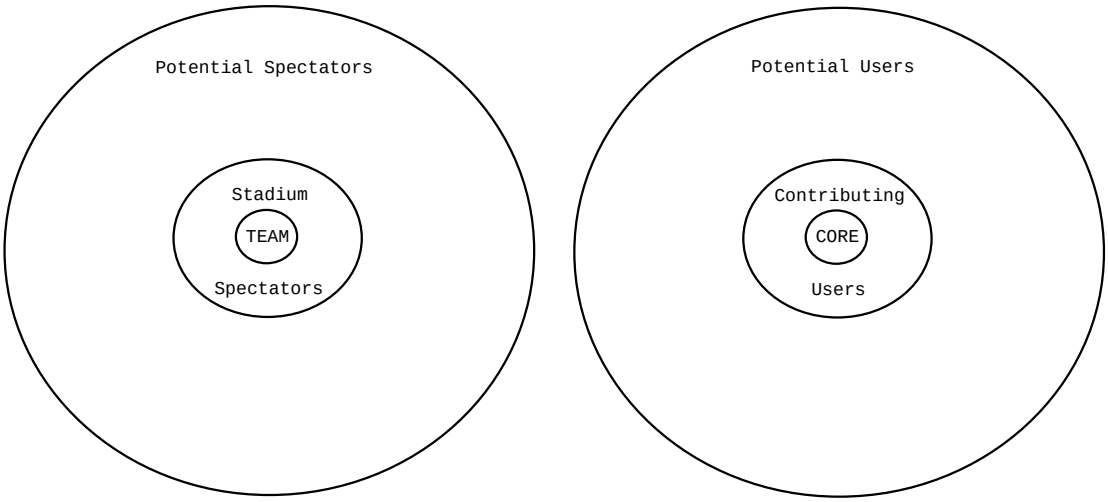


Figure 4.2: A figure illustrating the gap between the core and the potential users/spectators

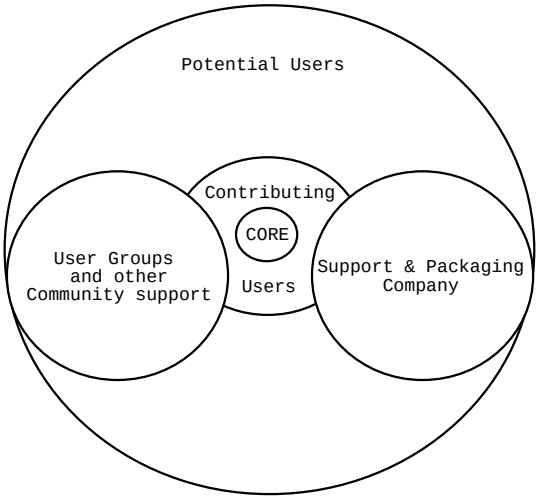


Figure 4.3: A figure showing how organizations is narrowing the gap.

4.4.3 Brands, Trademark and spin-offs

In a world where brands gains a steadingly increasing importance, companies invests in the games and the players in order to be able to use their investments as trademarks and gain from their brands. This strategy which have been established in the world of sports for a long while, is now gaining a larger momentum in the world of Open Source. The most famous hackers are recruited by companies in order to build a brand both within the community as well as outside it. Hiring an established name is a way of saying to the community, - we come in peace, look, this guy understood that and you trust him, as well as using it in their marketing. As we saw in the previous section, the brand is a key component to building trust, which is why people buy your repackaged stuff rather than getting them for free from the Internet.

This kind of efforts could be described as what Veblen would call conspicuous consumption. Consumption with the purpose to illustrate your prowess. Vendor support for Linux started of as pure conspicuous consumption, a vendor gave away a small number of units of their product to a number of nerds in order to have them write software drivers for it. This was raised to another level as IBM committed to invest 1 billion USD on Linux related business.

4.4.4 Coopetition

The fourth type of business model is the one that defies the rule that Open Source business models is strongly related to sports. The term coopetition was originally invented by Ransom Love, CEO of Caldera, in order to describe what happened when a number of companies joined together in a Open Source project to cooperate on one module, but to compete on the final products. The main difference between this model and pure repacking is that companies are using the Open Source product as a (hidden) part of a certain product, rather than pure repackaging.

In the beginning of this paper I stated that companies should use Open Source when the gain from having the project open was won by the cost. This often happens when a company finds use for a hobby project, and decides that it is less costly to use the product created by the hobbyists and giving back their improvements rather than developing the whole product from scratch. This is what have happened to the projects studied in Appendix A, they were all once small hobby projects, before companies found the use for the products they created. Since then, the projects have converted into a loosely connected system of co-operation between the companies who are interested in the product.

As an example, the Linux Kernel is used in a large number of products such as firewalls, routers, refrigerators and cell-phones. The operating system which was originally developed by a Finnish student is now being developed by a large number of companies, rather than hobbyists. As I said, this is the business model that is unique for Open Source, even though companies could join together and develop a certain product, or part of a product together the costs associated with agreeing on the terms of the co-operation would be rather large.

Coopetition could be described as yet another step in the direction of specialization of corporations, outsourcing everything but their competitive advantages. Both IBM and Oracle is embedding the Apache web-server in their solutions, saving money which enables them to focus on their key areas. Even though coopetition might sound revolutionary, it is basically the same thing as utilizing third party modules in your software, which have been utilized in software development for centuries.

4.5 Open Source methodology

4.5.1 Brooks Law

Every universe has its laws. One of the most famous in software development is Brooks Law, stating that

> Adding manpower to a late software project makes it later. [3, p. 25]

This statement alone could be considered to be fundamental knowledge in project management, and not important enough to get a name from someone who did not invent it. However, Brooks further argues in his essays that developing software in large teams adds complexity which is larger than

the task itself. Therefore software must be developed in very small, tight-knit teams. The concept of "Mythical-Man Month" is that you cannot estimate the number of man-months a certain project will take simply by aggregating the time of each task, as the efficiency of a solitary mind is multitudes larger than a mind which is a part of a team.

The origin of the concept "Mythical-Man Month" also originates from one of the great Upanishads of the hacker culture, the story of the lonely hacker who sat down a month and rewrote a whole operating system from scratch. Or as Eric S. Raymond phrases it

> In the beginning, there were Real Programmers [6, p. 19]

The essence of Brooks statement is that all development efforts should be organized so that the number of minds involved in the programming decisions should be minimized at any cost, with a stab of assistants to help them out. When writers states that Open Source methodology has overproved Brooks Law, they are not referring to the law itself, but rather that development could be carried out not only with a large number of minds, but with minds scattered over the whole world.

Brooks law is still valid in Open Source projects

We have seen previously that most projects are single-man projects. Those project does not have the problems Brooks described. As they occupy only one mind, there are no coordination costs. Projects with a large number of developers are often governed with a large number of hierarchical levels, with one or a few developers at the highest level. This leads to the fact that most decisions are made at the head level, by a small level of minds, in full accordance with Brooks Law. Therefore, one cannot, based on this, say that Brooks Law is no longer valid.

4.5.2 Bureaucracy

As I stated above, Open Source projects take on a strict hierarchical organization in order to avoid the effects of Brooks Law. Different Open Source projects have different ways of handling this, but when the project reaches a larger number of contributors, a core group is singled out, either formally or informally to be able to cope with the situation. The development of the Linux Kernel is structured in a 4 level hierarchy.

Level 1 Linus Torvalds

Level 2 Linus's Lieutenants

Level 3 Module maintainers

Level 4 Developers

Torvalds acts as the chief maintainer, having the final word on what is to be included and not, as well as releasing the official versions of the kernel. Each module, such as various file-systems, hardware drivers etc. has a separate maintainer [44], who are coordinated by the middle-layer, referred to as lieutenants.

The lieutenants are people who Torvalds trusts to a very large extent, they are responsible for collecting changes from the module maintainers, reviewing them and and acknowledging that they may be included in the kernel. Lieutenants are also responsible for maintaining bug-fixes for older versions of the kernel. This middle layer of abstraction were introduced in early 2002, when people started to complain that Torvalds were unable to catch up with his work [20].

Large Open Source projects are often hierarchical and bureaucratic

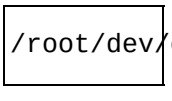


Figure 4.4: A figure showing the hierarch of the Linux Kernel project

In order for a infamous developer to get a feature into the kernel, he must find the correct maintainer, convince him not only that he have done a great job fixing the bug or adding the feature, but also that the maintainer should push it up to the lieutenant and try to convince him that this is so good that it should survive to yet another level.

The strong minded reader notices that this is in fact a formal hierarchy, designed in accordance with the ideas of Brooks.

4.5.3 Efficiency

Efficiency is a measure indicating how well we utilize the resources available. In the area of economics, we often choose to divide efficiency into multiple types of measures, a common way is to divide efficiency into external and internal efficiency. Internal efficiency is what we traditionally think of when discussing efficiency, it measures how much we are able to produce with the available resources. External efficiency measures a wider perspective, whether we are producing the right products.

External efficiency in Open Source is provided by utilizing the most basal properties of Darwinism. Even though the community is well connected and indexed, giving us the means to ensure that there are no redundancy[12], developers choose to create it. When you are in the need of a piece of software, perhaps a word processor or a piece of software to share Warez over the Internet, you have two choices. Either you use an existing piece of software, or you start a project of your own. Even though it often is multitudes easier to use an existing program, people choose to create their own.

If you look upon the Open Source communities as a company, providing products, this seems ludicrous, compare it to a company reasoning; we have no idea what our customers want, so we will produce one of each and see which they buy. On the other hand, if you look upon it as the anthill, it seems more wise for the ants to spread out, searching for food in different directions.

Open Source breeds diversity, not a single winner

This time, the ant-perspective seems more appropriate, there is no external efficiency in the Open Source communities as the developers themselves only acts on their own will, not aiming to meet any need of an external actor, such as a customer. To push the analogy a bit further, even though the ants are most likely unable to observe that they have no external efficiency requirements or that they are lacking directions by a wise leader, the sum of their activities creates an environment that provide them with the resources they need in order to survive. The efficiency of Open Source could be described as an anthill, a large number of individuals, serving their own interests, creating something that appears to be engineered.

Internal efficiency measures how you use your resources in order to produce as much of what you are producing, with as few resources as possible. How does this work on our anthill? Basically, everyone adds the parts which they need in order to make the software to fulfill their needs. If all developers working on a project share the same goals, Open Source development could be as efficient as traditional development, depending on how efficient the tasks are divided up to the appropriate persons. If people have fundamentally different visions of how the program is supposed to work, or how it should be achieved, a high level of efficiency is unlikely. Even if people agree on what they should do, many people might compete on the the best implementation. One example is the scheduler[13] of the Linux Kernel, a kernel only needs one of them, but they are a large number of implementations available as it is a prestigious task. Again we have Darwinism rather than efficiency, which obviously seems like a waste of resources to many.

Open Source is not necessarily an efficient way to develop software

> People mistakenly say "open-source software works because the whole Internet becomes your R&D and QA departments!" In fact, the amount of talented programmer effort available for a given set of tasks is usually limited. This, it is usually to everyones interests if parallel development efforts are not undertaken simply because of semantic disputes between developers. [6, p. 161]

4.6 Lock-in effects

I stated earlier, that lock-in effects is one of the single most effective ways for companies to ensure that their customers will return, no matter on how satisfied they are with the product itself.

Let us try some linear formulas, a favorite among economists. To make it a bit more interesting, I will assign some letters to the formula. The new product costs P_{new} in license, $P_{hardware}$ in

[12] If I need a certain piece of software, a few quick Internet searches provides a full overview of whether a solution exist or if someone is working on one. This is the benefit of the "Release early, release often" policy. This enables developers to avoid redundancy if they want, in comparison to academia where it is hard to find papers even after they are published.

[13] As there often are a number of programs running at the same time in the operating system, sharing a single CPU someone have to decide which program to be run, this is the work of the scheduler. By switching the capacity of the CPU between the programs the user is given the impression that multiple tasks are carried out simultaneously.

purchase of new hardware and $P_{install}$ in basic installation. Upgrading the old platform costs U_{old}. Then we have to consider the legacy effects. Converting all old data costs $C_{data}$and retraining all staff on the new system costs C_{staff}.

The formula rewritten becomes:

$$P_{new} + P_{hardware} + P_{installation} + C_{data} + C_{staff} \leq U_{old}$$

In the traditional non-Open Source scenario, all those costs are existent and significantly larger than zero. The absolute and relative sizes of the costs varies depending on what scenario we are studying. If we have a supercomputer, we have large investments in non-standard hardware, a very small number of staff to train to the new system, but a larger number of home made software which might need to be rewritten to the new platform. In most non open-source cases, the left side is multitudes bigger than the right side of the formula, and this is the main reason why there are large amounts of money to earn for after-sales consulting. Some software vendors such as SAP[14] sells a huge, complex system, which cannot be started without millions spent on customization of the system itself. And as soon as the system is purchased, the consultants lives happily for ever after. With good knowledge of the formula stated above, they price their services not on the actual required work, but on the pain threshold of the buyer.

Let us consider the ultimate Open Source scenario, in order to estimate how well we may avoid the lock-in effects. In the best-case scenario, we have no license costs, we are able to reuse our current hardware and in-house resources are able to install the software themselves.

This leaves us with:

$$P_{installation} + C_{data} + C_{staff} \leq U_{old}$$

In most cases, a company would probably chose to buy the product from a vendor specializing in Open Source, where the cost of license purchase would not be zero, but in this scenario, we take the luxury of assuming that we are in fact able to download the software from the Internet for free. Even though the installation is carried out in-house, it comes with a cost.

Ignoring the practical issues that we have to run the two systems in parallel for a migration period, and therefore most often are unable to use the same hardware even though the software might be compatible enough to run on the same platform. We are still left with what would be the two largest costs, converting the legacy data and training the staff on the new software.

If we are trying to switch from Microsoft's Office Suite, into another word processor/spreadsheet package, every document we have created through history must be converted in order to be able to use them in the new system. $C_{data}$is often the single largest cost of systems migration.

There are three common misconceptions when arguing that Open Source avoids lock-in effects:

1. Open Source systems minimizes C_{data}
2. The cost of retraining staff is small / zero
3. Open source systems have lower Total Cost of Ownership (TCO)

It is hard to reach any general conclusions regarding the total cost of ownership as it is dependant on too many parameters to be discussed. There are investigations showing that Open Source systems have lower TCO, and there are contradicting reports, much depending on who funded it. It is hard to generalize software systems to a standard scenario, which is why I will not discuss this point further in this text. The other two points however, will be discussed in this section.

Proprietary systems often comes with proprietary file formats, which could make it harder to convert data from one system to each other. Open source system could be designed to minimize the cost of converting data to other systems, yet again, so could proprietary systems. What Open Source system guarantees is that it is possible to do the conversion, the file format could be reconstructed by reading the source code, with the appropriate resources available. It does not however, guarantee

[14]Number one accounting systems for Fortune 500 companies.

that it will be easy. Therefore Open Source in it self provides only the most basic level of protection against lock-in effects.

Program	*# file formats*	*Open Source*
AbiWord	18	yes
Lyx	5	yes
Microsoft Word	25	no
Open Office	5	yes

Table 4.6: Number of file formats various word processors can export to.

My copy of Microsoft Office is able to save in 25 file formats, my three Open Source word processors, Lyx is able to export to 5, Abiword 18 and Open Office to 5 different file formats. As you can see, the super-proprietary Microsoft Word, is not all that proprietary after all.

Open Source is no guarantee for reduced lock-in effects

The cost of retraining staff is often ignored or forgotten when discussing the savings of relying on Open Source software. As shown previously, most Open Source projects are not targeted end-users, yet most articles tend to describe Open Source as a potential competitor to manufacturers of end user software. If the software to be switched is not an end-user system, this cost is often very small in comparison with the cost of converting legacy data. On the other hand, the silent knowledge existing in the organization, on how the software should be handled when something fails, often takes years of failures and experiences to create. I have never seen the cost of relearning this "experience" of system administrators estimated in a financial proposal, neither on proprietary nor on open systems. In software projects where there is an end-user interfacing the system, the cost of tutoring is huge.

Even when considering first deployment of a certain system, the cost of retraining staff is existent. The workers previous experience with similar systems gives them an advantage which often could be benefited from when deploying standard system compared to customized ones. Even though investigations have suggested that certification such as the EDCL should be created for Open Source products as well as proprietary systems [41], this have not been realized[15].

4.7 Participation in commercial Open Source projects

We have discussed a bit about motivation in hobbyists Open Source projects. This section will discuss a bit further what might happend if a company chooses to open up their product in order to get a number of developers to help them out with the development.

In order for a gift to be appreciated, it must contain some kind of value for the recipient. If you give someone a artifact that they have use for, they will use it, and possible, improve it if there is something that they may do in order to make the artifact solve their problem in a more efficient way.

The previous sentence contains a number of weak points, where there is a potential for a failure. First off, the artifact given to the community, must have a value for the community which is supposed to contribute to the project. Further on they must be able to understand programming in general, and the source code in particular. There are three main methods to get someone to contribute to a certain project:

1. The license requires you to contribute if you make improvements
2. There are monetary (or other kind of external) motivations available for contributors.
3. There exists a community where people want to be members, and contributing is a requirement to belong to the community

In order to get thousands of developers to develop your software, you must provide either of these three motivations, they could be used either as a combined way or only one of them.

[15]The suggestion in itself is rather sound, but the context in which it is given it gives the impression that the author have misunderstood both the concept of EDCL and Open Source.

The myth that an Open Source product attracts developers relies on the assumption that you easily could provide the community of method number three, something which only a few hobby projects have been able to accomplish. I have not been able to find one single project where a commercial entity have opened a commercial software project in a way so that method three have been fulfilled. The project which have come close is the Mozilla project which will be described in the next chapter.

One of the most common ways for companies to engage in Open Source activities is to open supplemental products, around their core product. This is a strategy which is utilized by for example Oracle, who tries to lower their support efforts by releasing certain non-core products as Open Source in order to get help with the development and support efforts. Even though this might seem like a sound decision, as the products does not give any direct earnings, there should be no loss by giving them away. This, of course, assumes that there are no costs maintaining an open software project.

There exists a number of potential pitfalls when opening a commercial products.

Opening your source does not automatically lead to a large number of contributors

If someone pays a vendor large amounts of money for a product and the support services covering it, they expects the vendor to take care of their problems, rather than contributing to an Open Source project themselves[16]. After all, they have paid money in order to reduce the insecurity which could be argued is associated with Open Source projects, hence, they expect the vendor to do exactly that. If you could save the license fees by using a free Open Source product, you are more likely to spend a fraction of that cost on improving the product so that it meets your needs. To return to the Evian metaphore presented previously; if I have paid for Evian, I expect it to come in bottle, rather than having get it from my tap.

Michael Tiemann describes what happened when his company took over the maintaintance of the GNU compiler, suddenly, everyone expected that problems would fix themselves.

> Tensions rose between sales and engineering while the Open Source model seemed to be working in reverse: the more development we did on GNU software, the less we got back from the Net, until we were doing over 50% of all GNU toolchain development. [6, p. 82]

Further, in order to have a large number of contributors to a project, you naturally need to have a large number of developers who uses and are familiar with the product, who for some reason, are interested to dedicate time on the project. Opening products which requires the purchase of a another, often expensive, product limits the number of potential developers to the number of customers.

As shown previously, most of the Open Source projects targets developers, not end users. This is because there is a often hidden assumption that the users of a certain project are themselves developers. If you have a product targeting developers, you are more likely to get cooperation, than if you are targeting computer illiterates. If the users are unable to read and understand the source code, they are unable to contribute with fixes and improvements.

This is an internal inconcistency in the discussion about the fact that everyone can participate and fix bugs in Open Source Software. On one hand, people argue that programming is an art, the creation of software is a highly artistic thing for which you have a talent. On the other hand, people argue that everyone can be developers, reading and fixing source code as they use their programs.

The popular quote from Torvalds that:

> "given enough eyes, all bugs are shallow"

is valid in his limited context. If enough people uses the code, and learns how it works, most of the bugs will be ironed out, as the bug likely will occur to someone, which will the fix it. The problem is that this is only valid for such a special context as a operating system kernel, and even with such a specialized software, it is only valid for the inner circle of people developing the kernel.

When developing software for developers, the statement have some validity, and when you have multiple companies interested in the same product, there will be more eyes from the developers of the different perspectives of each company. The main benefit for the Linux users is not that they are able to modify the software and fix bugs, most of them are unaware about what is running under their hood.

[16] Oracle database products begins at approx. $30,000 ranging up to several millions.

Chapter 5

The Concept of Open Source exemplified - The Mozilla Project

Few things are harder to put up with than the annoyance of a good example.

– Mark Twain

The purpose of this chapter is to illustrate how a Open Source project fails

5.1 The story of Netscape Communication Inc

Almost exactly ten years ago, Marc Andressen wrote a humble message to the mailing-list WWW-Talk, the forum for anyone who were involved in the new www-standards.

I'd like to propose a new, optional HTML tag IMG.

When Internet caught on, we learned that it had been there since the 60s, a result of cold war efforts to keep computer networks up even though one node were brought down by the enemies, but it simply never mattered to us before. In much the same way, the standards enabling us all browse the web were formalized over ten years before this story begins. The story of web browsing did not really matter before the day Marc Andreessen joined the party and decided not only to write his own web-browser, but to enable it to show pictures and other multi-media content.

Mosaic 1.0 were released on 21 April 1993, this was when the world as the youth knows it began, if you are elder than that, write the date down to educate your kids about the world before the Internet.

Remember, Internet had been at many campuses for years, companies had e-mail, the technical infrastructure were already there so the WWW were able to catch on quicker than anyone might have predicted, as soon as it could be illustrated to the masses through on-line pictures of a coffee-pot in Cambridge[1].

Microsoft surely missed that it was coming, Edstrom and Eller suggests in "Barbarians Led by Bill Gates" that Microsoft were to focused on high-bandwidth issues as Gates were building his new, connected, home [8]. This should be taken by a grain of salt, but the truth remains that Microsoft were totally uninterested in Internet before 1995. One who did see what was coming, or who made sure that it did come was Jim Clark. After being bored with his supercomputer company Silicon Graphics Inc. he was looking for new ventures when he stumbled over Marc Andresseen. They teamed up and started Mosaic Communications Corporation in order to make money on what Mosaic had started.

[1]The Trojan Room Coffee Machine were the first web-cam in the world. It was set up at Cambridge University so that the staff could check if there were fresh coffee in the pot without leaving their desks. When Internet started to gain media coverage, there was not much else to show, this was the most hip thing you could do, and they attracted millions of visitors.

Mosaic were later renamed to Netscape and continued to develop their browser, which were sold to corporations but free for students and non-profit usage. Netscape also made money on selling the servers delivering the web pages to the browser. By controlling both the client and server Netscape were able to add features to the system without going through a long standardization process first. As they were holding a de facto monopole on the browser market at that time, they were able to sell products without much overhead, the company was immature, but so was the market. The heroistic biography of the Swedish web-site Torget illustrates the frustration on using their products, not properly adjusted for international characters and different taxes [14].

When Clark needed more money to build the boat of his dreams he decided to go public with what had become Netscape and it became the first IPO of the Internet era, the bubble of the new economy had started as early as 1995 [22].

If Microsoft had missed the opportunities before this point, they acted rapid to adjust to the new circumstances. Few times before in history have a large company turned the ship that quick. The first version of Internet Explorer Microsoft purchased from SpyGlass Inc., the company holding the rights to the old versions of Mosaic source code originally developed at NCSA. The feature set were highly reduced, it is my honest belief that the only use of that browser was to download a never version, but it was done quick enough to bundle it with the upcoming version of Windows.

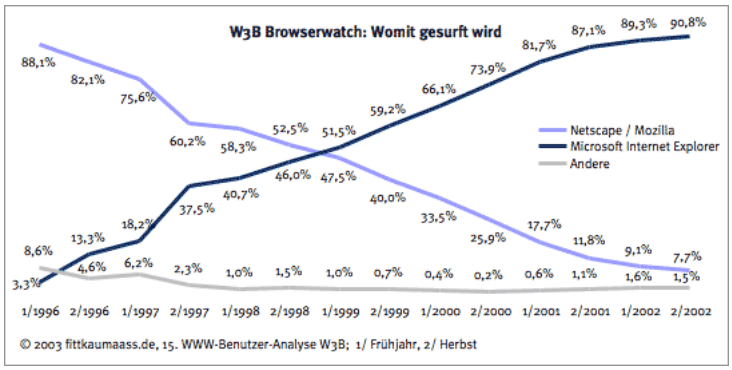


Figure 5.1: A figure showing Web Browser market share

Figure 5.1 illustrates what happened when Microsoft allocated every resource available to succeed with their brand new Internet strategy. The browser war began, and the trend were evident early. Microsoft's browser were tightly integrated to the operating system, making it quicker. Later versions of the Windows operating system is so knit in with the browser that everything from the desktop to file-browsing windows really are instances of the browser, simply type in the web-address anywhere and you will be redirected to the right web-page. It is hard to compete with that.

Jamie Zawinski, Netscape employee number 20 suggests that the result of the browser war is not only due to the massive power of Microsoft but that Netscape lost their power to innovate as the company grew bigger.

> When we started this company, we were out to change the world. And we did that. Without us, the change probably would have happened anyway, maybe six months or a year later, and who-knows-what would have played out differently. But we were the ones who actually did it. When you see URLs on grocery bags, on billboards, on the sides of trucks, at the end of movie credits just after the studio logos – that was us, we did that. We put the Internet in the hands of normal people. We kick-started a new communications medium. We changed the world.
>
> But we did that in 1994 and 1995. What we did from 1996 through 1999 was coast along, riding the wave caused by what we did before. [47]

5.2 The birth of the Mozilla project

By early 1998, the trend became rather obvious. Netscape had not only lost half of it market share on the market place, they were also attacked on the server side, where their revenues were found. Microsoft servers came with a easy configurable Internet Information Server for free, together with Open Source alternatives such as Apache. Netscape were stabbed where it hurt them the most. Something had to be done, if nothing radically different happened Netscape would ultimately go broke. Finally, Frank Heckner, a senior Netscape employee whose internal papers earlier had played an important role for the company made a suggestion, so crazy that it might just work. The paper titled "Netscape Source Code as Netscape Product" analyzed the case for the company and suggested that Netscape should be releasing their browser as Open Source. The exact contents of this paper have not been released outside Netscape, so we can not be sure exactly what he said, but it did convince Netscape executives to be bold, or desperate, enough to make the move.

On 22 January 1998 Netscape announced that they had made the decision to Open Source their product, the Netscape Browser, commonly known simply as Netscape [28]. In interviews following the press release, Netscape cited the work by Eric S. Raymond, the Cathredal and the Bazaar as an important inspiration for the decision.

This Open Source project became named Mozilla, after the internal name of the browser, in order to separate the project from the company itself, and a separate web-site and development team were set up [29]. The Mozilla project was the first commercial product to be released as Open Source by a commercial company, it became the 1998 flagship of the Open Source movement which gained its largest momentum right at that time. This were the early days of todays Open Source movement, the time when the concept reached the press, and the movement took on.

Up until today, Mozilla still is the best known example of an opened product.

5.3 The prerequisites of the Mozilla project

From an Open Source perspective the Mozilla project had everything rolled out right before its feet. It was a project which gained much attention, both from the press and the community itself. Members from varioush Open Source communities were invited to discuss not only how the project should be organized, but how to agree on a license everyone could accept [30]. It had a large number of users, there were many potential developers as the application ran on many platforms and most developers actually uses a web browser on a regular basis. It even had the largest trump of it all, the success of the Mozilla project would not only be the ultimate proof that the concept of Open Source were viable as a way to for commercial companies to make business, it would also kick Microsoft in the but, something Open Source developers often tend to like.

Releasing the source code was not a small task. Most development were halted in order to have every resource to help make it happened. There was much work to be done before the public could be allowed to see the source. Each line of code had to be reviewed, comments removed etc, and there were millions of lines of code. A big problem was that Netscape were using a number of third party modules, without the code would be unusable

> One of the largest issues was the disposition of the third-party modules included in the browser. Communicator contained over seventy-five third party modules in its source, and all the code owners needed to be approached. [6, p. 199]

When the modules were removed, the government did its best to add to the workload of the engineers:

> The removal of the cryptographic modules was another tremendous task for the engineering team. Not only did the government insist that all cryptographic support had to be removed, but every hook that called it had to be redacted. [6, p. 200]

Even though there were early attempts to reinsert crypto code, it was not until March 2000, nearly two years later, the crypto code were reinserted into the standard development tree, available for testing [40]. The browser is usable even without crypto code, but it makes it impossible to visit encrypted pages such as banks or secure purchases at e-retailers.

Finally, after three months of hard work, the source were released on the 31st March 1998. The lizard were finally free, and there was much rejoicing.

5.4 The failure of the Mozilla project

Shortly after the code was released, the problems began to rise on the horizon. There had been almost no development at Netscape the first three months of 1998, simultaneously as Microsoft were going full speed ahead Netscape were stressed to get the next minor version of their browser out on the market. The browser war were not over yet. The task to get the released source code to run well was a too huge task as there were plenty of holes in the code due to removed modules. Therefore they continued to work on an older code branch, pulling most of the resources from the Mozilla project onto getting Netscape 4.6 out of the door. The first important months, the Mozilla project were an empty shell of a few project managers and millions of lines of unusable code.

Exactly a year after the initial release of the Mozilla source code the head engineer Jamie Zawinski quit the project as it was not progressing as he would have liked to. His rather bitter resignation letter [47] is a very interesting read for anyone who is interested in problems associated with opening products. He concludes that there have been little outside involvement in the project and concludes that the released code really was not usable for a external developer.

> People only really contribute when they get something out of it. When someone is first beginning to contribute, they especially need to see some kind of payback, some kind of positive reinforcement, right away. For example, if someone were running a web browser, then stopped, added a simple new command to the source, recompiled, and had that same web browser plus their addition, they would be motivated to do this again, and possibly to tackle even larger projects.
>
> We never got there. We never distributed the source code to a working web browser, more importantly, to the web browser that people were actually using. We didn't release the source code to the most-previous-release of Netscape Navigator: instead, we released what we had at the time, which had a number of incomplete features, and lots and lots of bugs. And of course we weren't able to release any Java or crypto code at all.
>
> What we released was a large pile of interesting code, but it didn't much resemble something you could actually use.

Finally, after spending three months cleaning up the code so that it could be released, spending more months developing old code in order to push out a new release of the old browser, the result was not only a slip by at least six months, but a resulting Open Source project with code nobody understood how to use. A piece of software that was increasingly hard to maintain and update in order to support new standards. The code was ripped away, and work began both on a new rendering engine and new interface code, basically a total rewrite of the whole browser.

Early 1999, AOL acquired Netscape, a merge which questioned the future of the project [39]. As Mozilla was no keystone in the business strategy of AOL, and the schedule were already slipping, they considered killing the project off from the start. Despite the fact that they kept the project running, the AOL merge, and later on the merge with Time Warner put Mozilla further and further away from the core business of the main funder.

Shortly before Mozilla's first birthday Netscape had to lay off 850 workers. On its birthday, Jamie Zawinski, project leader left the project, due to the slow progress. During this time, many talented people left Netscape, and many more got fired. A year after the project were started they had reached the point where they had thrown out all the old pieces and started putting new pieces into work, people who had been reassigned to the Mozilla project from the work on legacy code were leaving, and the project yet again became drained.

In August 2000, the license were changed into a dual licensing model, using both the Netscape specific license as well as the Gnu Public License, later on, a third license option the Lesser Gnu Public License were introduced. The reasons for changing license was both that there was a lack of developers from outside of Netscape, as well as to meet some criticism from the Free Software Foundation [11] that the original license was too restricted to be called free [26].

Mozilla 1.0 was released 12th June 2002, over four years after the magic April fools day in 1998, and there was some rejoicing.

5.5 The epilogue of the Netscape era

The decision that would give Netscape a lead in the browser war, by increasing their development efforts by opening the product created a huge slip in the schedule of development. A year after the lizard were freed, there were no sign of any new generation Netscape browser, and Microsoft had gained an even larger lead with their next major release Internet Explorer 5.0 [46].

Last year became a rather good year for Mozilla, the code were used to a number of new clients and the total market share rose up to 1.7% of the Internet users, stealing the third place from Opera. Mozilla developed as a platform for various network activities and books were published on how to embed Mozilla in various solutions, but as for now, the browser war is over.

Frank Heckner concludes on Mozilla.org web-pages that [12]:

> There is no "royal road" to open source for commercial software companies. To quote Jamie Zawinski, you can't "sprinkle [a project] with the magic pixie dust of 'open source,' and have everything magically work." To elaborate, a company cannot simply release source code, put a few newsgroups up, and expect distributed development to magically self-organize; it will need to make a sustained effort requiring various people's time and attention in order to get a distributed open-source development effort to the point where it can produce results.

Opening the product obviously did not help Netscape to win the browser war. What happened to the great efficiency of Open Source methodology?

At this point, shortly after Mozilla's fifth birthday, there are a number of Mozilla based projects. There are over 15 different browsers based on the source code that Netscape released the spring 1998, not one of them manages to compete with Microsoft on the market share, but a lot of developers got to build the browser they wanted. But then again as Tim O'Reilly states it [6, p. 194]

> Evolution breeds not a single winner, but diversity

Few of the outside developers who joined the Mozilla project had any reasons for following the agenda put out by Netscape, like ants they built their own solutions, walked their own paths.

Then again, if Mozilla had not been released as Open Source in 1998, who knows what might have happened, most likely the project would have been killed and hidden in a drawer long ago after some of the mergers and acquisitions the company have gone through. Studying the browser usage graph above indicates that the decision what a somewhat desperate attempt to win the war. They did not win the war but they secured the future existence of the product.

Mozilla is larger than Netscape, and evolves in different directions every day, but Netscape are unable to direct the evolution. The opening of the product clearly delayed Netscape at the most critical moment, Many have created efficient solutions based on Mozilla, but Netscape is not one of them.

Chapter 6

Conclusions

All generalizations are false, including this one

– Mark Twain

The purpose of this chapter is to summarize what conclusions I have reached and give pointers to further potential research areas.

6.1 The myths revisited

Brooks law states that software development must be carried out in such a way that there are a minimum of brains involved, as the efficiency of programmers decreases the more people having to agree on different things. Open Source project has been considered a proof that Brooks was wrong, that a large number of people are able to create great software. I have shown that Open Source projects either is so small that the problem Brooks addresses is not applicable, and that large projects are organized in accordance to brooks theories on how to minimize the number of brains involved in each task.

- **Conclusion:** Brooks law is still valid in Open Source projects

In contradiction to what one might have assumed from reading articles about Open Source, most of the projects fails. Project often dies at early stages, before releasing any stable products, without building a sustaining community of developers and users.

- **Conclusion:** Most Open Source projects are failures

The Open Source culture is often dated back to the early 70s, where people working in MIT labs and other places shared values of freedom. Even tough the few elderly who are still active in the community often emphases their work, most developers joined and formed the culture and communities around Open Source during the 1990s. We showed that only a fraction of todays hackers had freedom as a primary motive.

As many papers are based upon the works stating that the culture of todays Open Source is derived from the older hacker culture of the 70s, we must be careful on what false conclusions may have been drawn from this assumption.

- **Conclusion:** Open Source culture is a product of the 90s, not the 70s

There are many costs involved with switching between different systems, only a minor part of them could be reduced by using Open Source software. The major costs such as installation, data migration and staff retaining remains even though Open Source components are used. Open Source guarantees that it will be possible to convert the legacy data in the system to another system, but it does not assure that it will be easy.

- **Conclusion:** Open Source is no guarantee for reduced lock-in effects

The era of the new economy featured nerds on the front page on high profile magazines such as Fortune, Time Magazine, Forbes etc. The story of a group of nerds terrifying "big bad Microsoft" is still a easy way to write a story, the David versus Goliath story. Evidence shows that in large projects where there are commercial interests, up to 70% of all development efforts are carried out by professionals employed by the companies benefiting from the product.

- **Conclusion:** Our most famous Open Source projects are not built up by nerds working for free, but professionals, employed by commercial companies to contribute to the projects.

The beauty of Open Source should be that anyone could participate, and that It didn't matter whether the solution came from a Ph.d. or a high school student. On the contrary, evidence were presented that large Open Source Projects are built up in an hierarchal fashion, and your solution is more likely to be accepted the higher you are in the hierarchy. The maintainer is the one who has the final word, and his trustees are more likely to get their will through.

- **Conclusion:** Large Open Source projects are often hierarchal and bureaucratic

The Mozilla project taught us that, despite good preconditions, there is not necessary a large number of people interested or competent enough to contribute to your project. There are many obstacles in creating a successful Open Source project.

- **Conclusion:** Opening your source does not automatically lead to a large number of contributors

I showed both in our section about efficiency and in the Mozilla example that Open Source efficiency works as an anthill, where all the actors use a certain project to fulfill their own interests. Open Source breeds diversity, not a single winner. Perhaps this is why Open Source could be beneficial as an innovator, as some ant is likely to travel a path which otherwise should not have been evaluated.

Open Source is not efficient as a way for a company to develop a project in a certain direction, if they are not willing to put in all the work themselves in order to make sure that it evolves the way they want.

- **Conclusion:** Open Source breeds diversity, not a single winner

With this in mind, it would be interested to know how Hewlett Packard have thought about how they are to direct their ants to create the products the company needs, solving the bugs that the customers wants. Companies organize themselves in order to make sure not only that there are no duplication of work, but that all tasks which needs to be carried out actually is. If organizing a company as an anthill would be efficient, how people started to organize their organizations at all?

- **Conclusion:** Open Source is not necessarily an efficient way to develop software

The investigation on Open Source projects showed that most of the projects targets the community itself. Most projects were targeted developers and systems administrators. Further they were targeted Unix systems rather than Windows systems. As there is no clear intent to give the fruits of their labours to total strangers, it might be premature to talk about altruism.

- **Conclusion:** Open Source projects often targets the community itself, rather than external actors

Even though a formal market study have not been carried out, most of the well known companies I have identified who engages in Open Source does so with rather traditional buisness models, often resembling models used in sports. The main business models is either to repackage the products to target new customers, or to use the products as third party modules in their own products.

- **Conclusion:** Companies benefiting from Open Source are often based on traditional business models rather than revolutionary visions

6.2 Suggestions for further research

Even though the areas of Open Source have been studied to a large extent, there are many areas left to explore. This thesis have been a general overview, there is potential to go deeper at virtually any point discussed here but I will give some more concrete suggestions. First, it would be interesting to investigate how the community reacts to the commercialization of the game, how is money affecting the motivational factors. Why does the minority of hobbyists in the largest Open Source project keep working for free, when up to 80 per cent of thier peers are being paid for the same work. Second, it would be interesting to put Open Source into a wider sociological perspective, how come the Open Source communities are able to survive and attract people when most other communities are on the verge on abandonment.

Bibliography

[1] The Boston Consulting Group (2002), *Hacker Survey,* http://www.osdl.com/bcg/

[2] Broil, M. (2003), *Why Do People Write Open Source Software?*, http://slashdot.org/articles/03/04/26/1417247.shtml

[3] Brooks, Frederic Phillips, Jr (1995), *The mythical man month: essays on software engineering.*

[4] Caillois, Roger (2001), *Man, Play and games*, The Free Press of Glencoe, Inc.

[5] Elias, Norbert (1978), *The History of Manners,* Urizen Books, New York

[6] DiBona, Chris, Ockman, Sam and Stone, Mark (1999), *Open Sources: Voices of the Open Source Revolution*, O'Reilly & Associates

[7] Elias, Norbert and Dunning, Eric (1986), *Från riddarspel till fotbollscup: sport i sociologisk belysning*, Atlantis, Stockholm

[8] Eller & Edstrom (1998), *Barbarians Led by Bill Gates*, Henry Holt and Company Inc.

[9] Eriksson & Wiedersheim-Paul (2001), *Att utreda, forska och rapportera*, Lieber ekonomi, Malmö

[18] European Commission (2002), *Free/Libre and Open Source Software: Survey and Study (FLOSS)*, http://www.infonomics.nl/FLOSS/

[10] Fink, Martin (2003), *The business and economics of Linux and open source*, Pearson Education, Inc., New Jersey

[11] Free Software Foundation (2003), *Homepage,* http://www.fsf.org/

[12] Heckner, Frank (1999), *Mozilla at one: A Look Back and Head,* http://www.mozilla.org/mozilla-at-one.html

[13] Huizinga, Johan (1950), *homo ludens,* Roy Publishers

[14] Hörnfeldt, Erik, Hansson, Mattias (1999), *www.torget.se - Den fantastiska historien om ett av världens mest framgångsrika internetprojekt,* Bonnier Icon Publishing

[15] IBM (2003), *Open source software*, http://www-3.ibm.com/software/info/topic/opensource.html

[16] Judge, Peter (2002), *Ballmer: We'll outsmart open source* ,http://zdnet.com.com/2100-1104-959112.html

[17] Karp, Stefan (2000), *Barn, Föräldrar Och Idrott: En intervjustudie om fostran inom fotboll och golf*, Umeå universitet

[18] Kelly, Lisa and Ranger, Steve (2002) ,*Whitehall vows to end proprietary lock-in*, http://www.computing.co.uk/News/1133883

[19] Kollock, Peter and Smith, Marc A. (1999), *Communities in cyberspace*, Routledge, London

[20] Landley, Rob (2002), *Linux-Kernel Mailinglist: A modest proposal – We need a patch penguin*, http://www.ussgiu.edu/hypermail/linux/kernel/0201.3/1000.html

[21] Levy, Steven (1984), *Hackers: Heroes of the Computer Revolution*

[22] Lewis, Michael (1999), *The New New Thing - How some man you've never heard of just changed your life*, Hodder & Stoughton

[23] Melian, Ammirati, Garg, Sevon (2002), *Building Networks of Software Communities in a Large Corporation*

[24] Moody, Glyn (2001), *Rebel Code: The Inside Story of Linux and the Open Source Revolution*, Perseus Publishing, Cambridge, Massachusetts

[25] Moore, J.T.S (2003), *DVD: Revolution OS*, http://www.revolution-os.com

[26] The Mozilla Organization (2003), *Mozilla Relicensing FAQ*, http://www.mozilla.org/MPL/relicensing-faq.html

[27] MozillaZine (2000), *Mike Shaver Leaving Netscape*, http://www.mozillazine.org/talkback.html?article=1078

[28] Netscape Communications Corporation (1998), *Pressrelease: Netscape Announces Plans to Make Next-Generation Communicator Source Code Available Free on The Net*, http://wp.netscape.com/newsref/pr/newsrelease558.html

[29] Netscape Communications Corporation (1998), *Pressrelease: Netscape Announces Mozilla.org, A Dedicated Team and Web Site Supporting Development of Free Client Source Code, http://wp.netscape.com/newsref/pr/newsrelease577.html*

[30] Netscape Communications Corporation (1998), *Pressrelease: Netscape Makes Draft of Free Source Code License Available for Review*,http://wp.netscape.com/newsref/pr/newsrelease579.html

[31] Olson, Michael A (2002), *A business case for open source*, http://news.com.com/2010-1078-901341.html

[32] The Open Source Initiative, *Homepage*, http://www.opensource.org

[33] The Open Source Initiative, *The Open Source definition*, http://www.opensource.org/docs/definition.php

[34] Pavlicek, Russel C. (2000), *Embracing Insanity: Open Source Software Development*, Sams Publishing

[35] Putnam, Robert D. (2000), *Bowling Alone - The Collapse and Revival of American Community*

[36] Raymond, Eric S. (2001), *The Cathedral and the Bazaar: Mustings on Linux and Open Source by an Accidential Revolutinary, Revised Edition*, O'Reilly & Associates, Inc., Sebastopol

[37] Reasoning Inc. (2003), *How Open-Source and Commercial Software Compare: A Quantitative Analysis of TCP/IP Implementations in Commercial Software and in the Linux Kernel, http://www.reasoning.com/downloads/Open_Source_White_Paper_v1.1.pdf*

[38] Rehn, Alf (2001), *Electronic Potlatch*, KTH INDEK, Stockholm

[39] Slashdot (1999),*AOL Cosiders Ending Mozilla?*, http://slashdot.org/article.pl?sid=00/08/16/2157245&mode=thread

[40] Slashdot (2000), *Mozilla With Crypto Code released*, http://slashdot.org/article.pl?sid=00/03/09/1052248&mode=thread

[41] Statskontoret (2003), *Öppen Källkod (2003:8)*, http://www.statskontoret.se/pdf/200308.pdf

[42] Torvalds, Linus and Diamond, David (2001), *Just for fun: the story of an accidential revolutionary*, HarperCollins Publishers Inc., New York

[43] Veblen, Thorstein (1994), *The theory of the leisure class*, Dover Publications Inc., Toronto

[44] Vlasenko, Denis (2003), *Linuk-Kernel Mailinglist: lk maintainers*, http://www.ussg.iu.edu/hypermail/linux/kernel/0304.2/0323.html

[45] Williams, Sam (2002), *Free as in Freedom: Richard Stallman's Crusade for Free Software*, O'Reilly & Associates

[46] Wired (1999), *Where's Netscapes new browser?*, http://www.wired.com/news/technology/0,1282,18549,00.html

[47] Zawinski, Jamie (1999), *resignation and postmortem*, http://www.jwz.org/gruntle/nomo.html

Appendix A

Open Source Development - Who is paying for free software?

Introduction and method

This document aims to investigate whether open source development (OSD) in large open source projects (OSP) is still a matter of a group of individuals doing development on their free time as a way to increase their knowledge and having fun. Or if the project, after reaching a critical size where large companies incorporates the product into their strategies becomes a loose system of cooperation between corporations instead.

In order to investigate this further I have collected statistics from the mailing-lists archives from five of the largest, and most well known open source projects. Namely;

- The Apache Web server, the single most popular web server used today with a market share over 60%.
- The GNOME desktop, an user interface for the X windowing systems, currently shipped with most of the large Linux distributions as well as the default desktop for Sun Workstations.
- The Linux Kernel
- The SAMBA server, a piece of software that enables Unix servers to act as Windows NT/XP/2000 file servers.
- WINE (Wine is no Emulator), a piece of software that enables Unix machines using X86 processors (Intel/AMD) to run standard Windows software

The projects have been singled out not only because they are well known even outside the open source community but also as they were once small projects which have grown so large that the projects now have become strategic parts of traditional companies such as IBM and SUN. Not only have they attracted the software giants of the valley, they have also nurtured a number of start-ups focusing on packaging the projects above and thus enabling actors outside the community to benefit from the efforts.

The study were carried out by collecting statistics from over half a million messages sent to the projects development mailing-lists. Open source projects is usually governed by the development mailing-lists, therefore the activity on the mailing-lists may act as an indicator of the efforts put down by single developers. As the culture have low tolerance from free-riders on the lists (people making promises but never delivering) the number of people delivering nothing but talk on the lists should be minimized. I do not state that this is the ultimate way to measure activity in open source projects, there are many others which may, or may not lead to more accurate results. I do however state that this method is good enough to single out the main contributors over time.

Making the classifications

After obtaining the names of the main contributors to the projects, I have tried to identify whether they are receiving financing by a commercial entity in order to obtain whether the project is ruled by commercial interests or if the project could still be classified as a hobby. I will not question the integrity of the developer obtaining the financial aid to continue working on the project full time instead as a hobby, neither will I try to assess whether the developer find themselves affected by the goals of their funder or if they feel that they are free to do what they want.

In order to make this work possible I have made some general distinctions in order to be able to classify the funders into black and white categories. A developer which is paid to develop on a certain project, is classified as a company contribution at all times. Even though there are some people out there not only contributing on working-hours but also on other parts of their projects on their free time, they are threated as paid on all their work. There are no distinctions done whether the company is actively encouraging the developer or if he simply gets away with doing free software work without the knowledge of his superiors.

During the data collection phases these data have been posted on the mailing-list and I have received many valuable comments and corrections on my data. One comment that I have disregarded is the common focus on what a contributor is. I have received a few comments such as; I wouldn't classify me as a major contributor, I just hang around and help users, I seldom write any code. This focus on writing code is perhaps natural in an environment of developers, but my experience is that there is a lot more into developing software than writing code. Therefore those people have been included, as we are not counting lines of written code, but the effort for the project and the community.

Validity of the data

Please do not read to much into the numbers presented here. Even though they are presented with an appearance of accuracy, you can not state that x.y% of a certain project is funded by company z. Focus on the patterns, the large trends. The goal is that you should be able to back up a statement such as "there is a significant amount of activity in project x which is funded by corporations, the main funders are y,z,w".

Acknowledgments

The world of Open Source Software is a wonderful place for a researcher. As the conversation is often carried out via mailing-lists, which are archived and publicly available, we are able to go back in history and follow nearly every sentence since the beginning of the project. Not only is the history archived carefully, the tools to index and process it is available right at my hands. I'd like to thank, not only the developers which made the research meaningful but also the people keeping the archives as well as the creators and teams behind some interesting software such as the Perl language, the MySQL database engine and the LyX editor for LaTeX. A special thank should also go to Mark Overmeer and the others behind the Perl-module Mail::Box which saved me some work in parsing the archives.

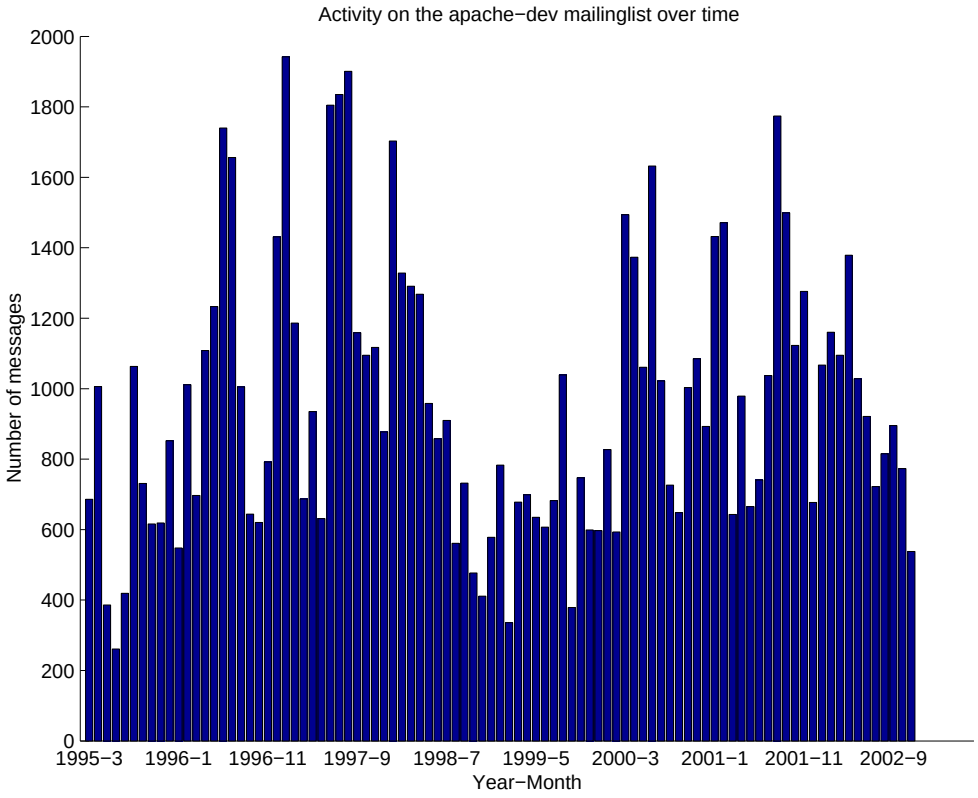
The use of trademarks have been use very respect-less in this text, use of a term in this paper should not be regarded as effecting the validity of any trademark or service mark.

The Apache Web Server

Background

According to the July 2002 Netcraft survey, over 60% of all Internet websites were powered by the Apache Web Server. Apache was once a small set of patches to the original NCSA web server but have grown to be a part of web-solutions from large companies such as IBM and Oracle. I have studied 89 502 messages in the apache-dev mailing-list between 1995-02-28 and 2002-12-17 in order to measure the activity.

Activity over time



Main development contributors during year 2002

# Mails	*Percentage*	*Aggregated*	*Person*
736	6.90 %	6.90 %	Justin Erenkrantz, Unknown / Self-financed
702	6.58 %	13.48 %	Jeff Trawick, IBM
676	6.34 %	19.82 %	William A. Rowe, Covalent Technologies
603	5.65 %	25.47 %	Ryan Bloom, Covalent Technologies
591	5.54 %	31.01 %	Cliff Woolley, Unknown / Self-financed
489	4.58 %	35.60 %	Aaron Bannert, Covalent Technologies
404	3.79 %	39.38 %	Brian Pane, Cnet
402	3.77 %	43.15 %	Jim Jagielski, JaguNET
398	3.73 %	46.88 %	Bill Stoddard, IBM
271	2.54 %	49.42 %	Greg Ames, IBM
262	2.46 %	51.88 %	Sander Striker, Unknown / Self-financed
254	2.38 %	54.26 %	Rodent Of Unusual Size, IBM
244	2.29 %	56.55 %	Joshua Slive, Unknown / Self-financed
241	2.26 %	58.81 %	Ian Holsman, Cnet
219	2.05 %	60.86 %	Graham Leggett, Unknown / Self-financed
204	1.91 %	62.77 %	Greg Stein, Unknown / Self-financed
158	1.48 %	64.25 %	Pier Fumagalli, Sun Microsystems
140	1.31 %	65.57 %	Dirk-Willem Van Gulik, Covalent Technologies
128	1.20 %	66.77 %	Bojan Smojver, Rexursive
121	1.13 %	67.90 %	Thom May, The Positive Internet Company Ltd
88	0.82 %	68.73 %	Brad Nicholes, Novell Inc.
83	0.78 %	69.50 %	Roy T. Fielding, Day Software Inc.
82	0.77 %	70.27 %	Jerry Baker, Unknown / Self-financed
79	0.74 %	71.01 %	André Malo, Unknown / Self-financed
77	0.72 %	71.74 %	Martin Kraemer, Fujitsu-Siemens Computers
76	0.71 %	72.45 %	Doug Maceachern, Covalent Technologies
69	0.65 %	73.09 %	Paul J. Reder, IBM
66	0.62 %	73.71 %	Sebastian Bergmann, Unknown / Self-financed
59	0.55 %	74.27 %	Stas Bekman, Unknown / Self-financed
57	0.53 %	74.80 %	Madhusudan Mathihalli, HP

Table A.1: A table listing the most active persons on the apache-dev mailinglist during 2002.

Main financial contributors during year 2002

# Mails	*Percentage*	*Aggregated*	*Company*
2542	31.86 %	31.86 %	Unknown / Self-financed
1984	24.87 %	56.72 %	Covalent Technologies
1694	21.23 %	77.95 %	IBM
645	8.08 %	86.04 %	Cnet
402	5.04 %	91.08 %	JaguNET
158	1.98 %	93.06 %	Sun Microsystems
128	1.60 %	94.66 %	Rexursive
121	1.52 %	96.18 %	The Positive Internet Company Ltd
88	1.10 %	97.28 %	Novell Inc.
83	1.04 %	98.32 %	Day Software Inc.
77	0.97 %	99.29 %	Fujitsu-Siemens Computers
57	0.71 %	100.00 %	HP

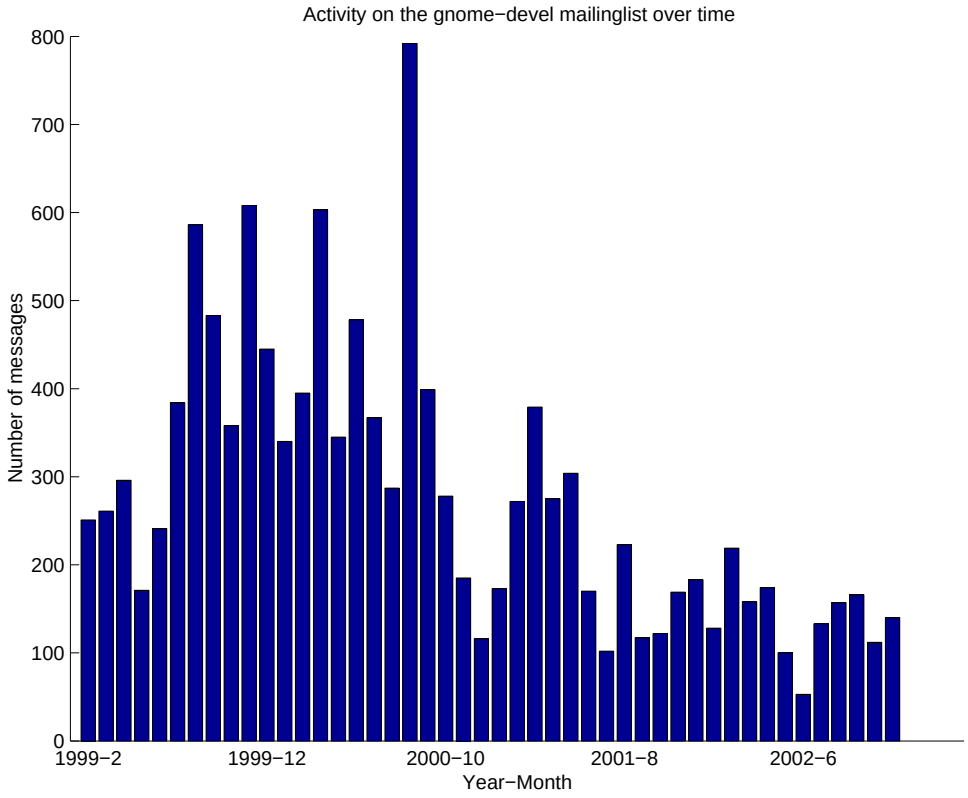
Table A.2: A table listing the most active companies on the apache-dev mailinglist during 2002. (Only persons in the previous figure is included.)

The GNOME desktop

Background

GNOME is the graphical user interface (GUI) project led by Miguel de Icaza. The project aims to create an easy-to-use interface to the X Window System used by most UNIX systems. GNOME is not only one of the most popular GUIs to Linux, it is also shipped as the default GUI on workstations from SUN Microsystems. In this study, I have studied 12 744 messages from the gnome-devel mailing-list between 1998-07-07 to 2002-12-16.

Activity over time



Main development contributors during year 2002

# Mails	*Percentage*	*Aggregated*	*Person*
105	6.64 %	6.64 %	Havoc Pennington, RedHat
64	4.05 %	10.68 %	Elliot Lee, RedHat
63	3.98 %	14.66 %	Michael Meeks, Ximian
47	2.97 %	17.64 %	Sean Middleditch, Unknown / Self-financed
44	2.78 %	20.42 %	Ali Akcaagac, Unknown / Self-financed
42	2.65 %	23.07 %	Cristiano De Michele, Unknown / Self-financed
36	2.28 %	25.35 %	James Henstridge, Unknown / Self-financed
30	1.90 %	27.24 %	Sergey V. Udaltsov, Unknown / Self-financed
26	1.64 %	28.89 %	Sander Vesik, Sun Microsystems
25	1.58 %	30.47 %	Philip Van Hoof, Unknown / Self-financed
24	1.52 %	31.98 %	Shane W. Clancy, Northrop Grumman
24	1.52 %	33.50 %	Kjartan Maraas, Unknown / Self-financed
22	1.39 %	34.89 %	Malcolm Tredinnick, Unknown / Self-financed
21	1.33 %	36.22 %	Jacob Berkman, Ximian
20	1.26 %	37.48 %	Miguel De Icaza, Ximian
19	1.20 %	38.69 %	Owen Taylor, RedHat
18	1.14 %	39.82 %	Michael Honeyfield, Unknown / Self-financed
18	1.14 %	40.96 %	Martin Sevior, Unknown / Self-financed
17	1.07 %	42.04 %	Allin Cottrell, Unknown / Self-financed
16	1.01 %	43.05 %	David Moles, Unknown / Self-financed
16	1.01 %	44.06 %	Franck Martin, Unknown / Self-financed
15	0.95 %	45.01 %	Mikael Hallendal, CodeFactory
15	0.95 %	45.95 %	Biswapesh Chattopadhyay, Unknown / Self-financed
15	0.95 %	46.90 %	Thomas Vander Stichele, Unknown / Self-financed

Table A.3: A table listing the most active persons on the gnome-devel mailinglist during 2002.

Main financial contributors during year 2002

# Mails	*Percentage*	*Aggregated*	*Company*
385	51.89 %	51.89 %	Unknown / Self-financed
188	25.34 %	77.22 %	RedHat
104	14.02 %	91.24 %	Ximian
26	3.50 %	94.74 %	Sun Microsystems
24	3.23 %	97.98 %	Northrop Grumman
15	2.02 %	100.00 %	CodeFactory

Table A.4: A table listing the most active companies on the gnome-devel mailinglist during 2002. (Only persons in the previous figure is included.)

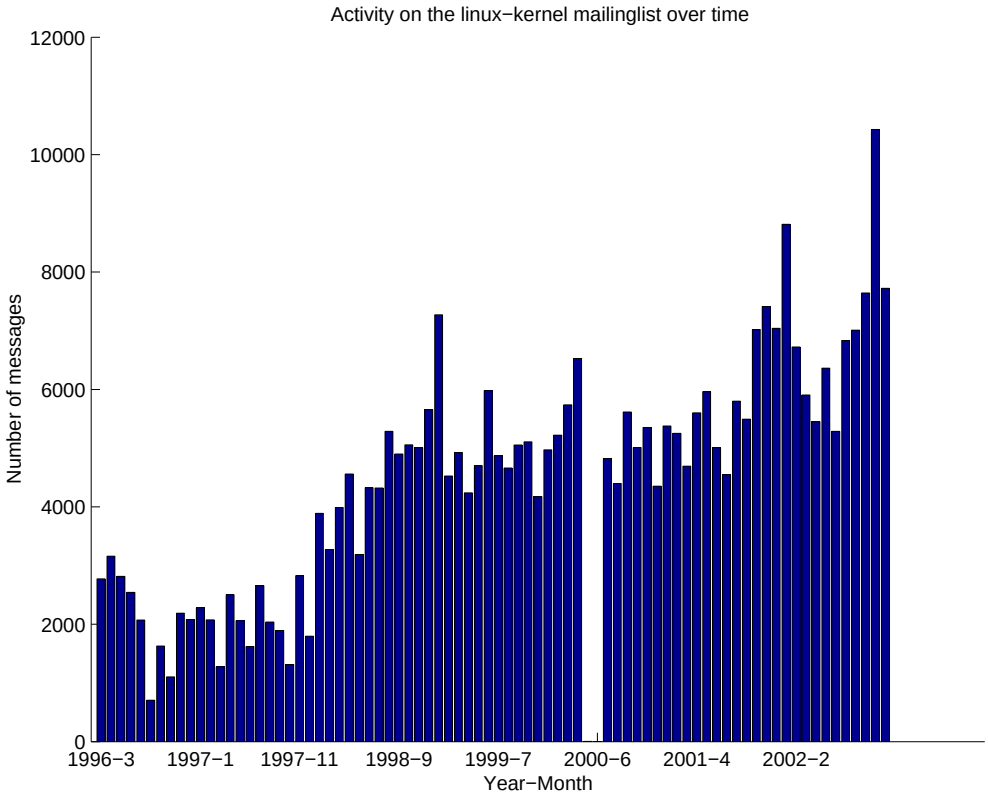
The Linux Kernel

Background

The Linux kernel is perhaps the most well known Open Source project in the world. Started, and still maintained by Linus Torvalds in 1991. Linux has become one of the fastest growing Operating Systems today, with millions of users over the world. Many companies use Linux as a large brick in their strategy, both as delivering products and services using the system as well as using it to

power its servers. This study covers 358 910 messages from the linux-kernel mailing-list between 1996-03-01 and 2002-12-25

Activity over time



Main development contributors during year 2002

# Mails	*Percentage*	*Aggregated*	*Person*
3561	4.27 %	4.27 %	Alan Cox, RedHat
1907	2.29 %	6.56 %	Andrew Morton, Moxi.com
1403	1.68 %	8.24 %	David S. Miller, RedHat
1343	1.61 %	9.85 %	Linus Torvalds, Transmeta
1143	1.37 %	11.23 %	Greg Kh, IBM
1076	1.29 %	12.52 %	Jeff Garzik, Mandrake Software
1035	1.24 %	13.76 %	Daniel Phillips, Innominate
934	1.12 %	14.88 %	Dave Jones, SuSE
931	1.12 %	15.99 %	Pavel Machek, SuSE
928	1.11 %	17.11 %	Rik Van Riel, Conectiva
877	1.05 %	18.16 %	William Lee Irwin III, IBM
873	1.05 %	19.21 %	Rusty Russell, IBM
869	1.04 %	20.25 %	Robert Love, MontaVista Software
864	1.04 %	21.29 %	Ingo Molnar, RedHat
756	0.91 %	22.19 %	Jens Axboe, SuSE
732	0.88 %	23.07 %	Martin Dalecki, Evision Ventures?
698	0.84 %	23.91 %	Andrea Arcangeli, SuSE
688	0.83 %	24.73 %	Christoph Hellwig, SGI
613	0.74 %	25.47 %	Andre Hedrick, Pyx Technologies
598	0.72 %	26.19 %	Bill Davidsen, TMR Associates
598	0.72 %	26.90 %	Russell King, Independant / Consulting for ARM Ltd
581	0.70 %	27.60 %	Alexander Viro, RedHat
576	0.69 %	28.29 %	Keith Owens, SGI
575	0.69 %	28.98 %	Zwane Mwaikambo, Unknown / Self-financed
558	0.67 %	29.65 %	Thunder From The Hill, Unknown / Self-financed
558	0.67 %	30.32 %	Martin J. Bligh, IBM
558	0.67 %	30.99 %	Vojtech Pavlik, SuSE
546	0.66 %	31.65 %	H. Peter Anvin, Transmeta
520	0.62 %	32.27 %	Richard B. Johnson, Analogic Corporation
471	0.57 %	32.83 %	Andi Kleen, SuSE
435	0.52 %	33.36 %	Roman Zippel, Unknown / Self-financed
434	0.52 %	33.88 %	Adrian Bunk, Unknown / Self-financed
430	0.52 %	34.39 %	Larry McVoy, BitMover
392	0.47 %	34.86 %	Andreas Dilger, Cluster File System
390	0.47 %	35.33 %	Eric W. Biederman, Unknown / Self-financed
378	0.45 %	35.78 %	Denis Vlasenko, Unknown / Self-financed
367	0.44 %	36.22 %	James Simmons, Transvirtual
363	0.44 %	36.66 %	George Anzinger, MontaVista Software
354	0.42 %	37.08 %	Adam J. Richter, Yggdrasil
346	0.42 %	37.50 %	Anton Altaparmakov, Unknown / Self-financed
345	0.41 %	37.91 %	Davide Libenzi, Unknown / Self-financed
334	0.40 %	38.31 %	Andries Brouwer, Unknown / Self-financed
299	0.36 %	38.67 %	Arnaldo Carvalho De Melo, Conectiva
285	0.34 %	39.01 %	Benjamin Lahaise, RedHat
282	0.34 %	39.35 %	Tomas Szepe, Unknown / Self-financed
280	0.34 %	39.69 %	Roy Sigurd Karlsbakk, Unknown / Self-financed
280	0.34 %	40.02 %	Randy.Dunlap, OSDL
279	0.33 %	40.36 %	Rob Landley, Unknown / Self-financed
272	0.33 %	40.69 %	Trond Myklebust, Unknown / Self-financed
268	0.32 %	41.01 %	David Woodhouse, RedHat

Table A.5: A table listing the most active persons on the linux-kernel mailinglist during 2002.

Main financial contributors during year 2002

# Mails	*Percentage*	*Aggregated*	*Company*
6962	20.37 %	20.37 %	RedHat
4908	14.36 %	34.72 %	Unknown / Self-financed
4348	12.72 %	47.44 %	SuSE
3451	10.10 %	57.54 %	IBM
1907	5.58 %	63.12 %	Moxi.com
1889	5.53 %	68.65 %	Transmeta
1264	3.70 %	72.34 %	SGI
1232	3.60 %	75.95 %	MontaVista Software
1227	3.59 %	79.54 %	Conectiva
1076	3.15 %	82.68 %	Mandrake Software
1035	3.03 %	85.71 %	Innominate
732	2.14 %	87.85 %	Evision Ventures?
613	1.79 %	89.65 %	Pyx Technologies
598	1.75 %	91.40 %	TMR Associates
598	1.75 %	93.15 %	Independant / Consulting for ARM Ltd
520	1.52 %	94.67 %	Analogic Corporation
430	1.26 %	95.92 %	BitMover
392	1.15 %	97.07 %	Cluster File System
367	1.07 %	98.15 %	Transvirtual
354	1.04 %	99.18 %	Yggdrasil
280	0.82 %	100.00 %	OSDL

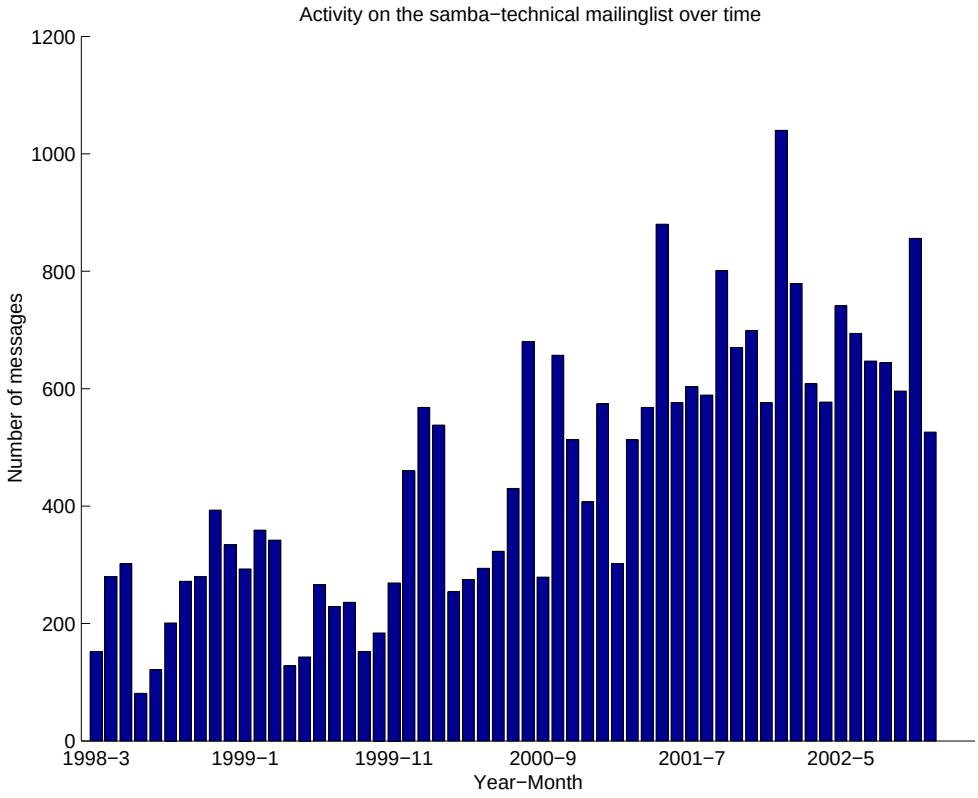
Table A.6: A table listing the most active companies on the linux-kernel mailinglist during 2002. (Only persons in the previous figure is included.)

The SAMBA Server

Background

SAMBA is a server enabling Unix workstations to speak the SMB protocol which is used by Windows 95/NT/2000/XP in order to share files. This software turns a Unix server into a full featured Windows server at not additional cost. A few years ago, an Irix (a flavor of Unix for SGI machines) were the fastest Windows File Server available. This study covers 26 052 messages from the samba-technical mailing-list between 1997-01-04 and 2002-12-17.

Activity over time



Main development contributors during year 2002

# Mails	*Percentage*	*Aggregated*	*Person*
636	7.95 %	7.95 %	Andrew Bartlett, Unknown / Self-financed
465	5.81 %	13.75 %	Gerald Carter, VA Linux
426	5.32 %	19.08 %	Richard Sharpe, Pansas
364	4.55 %	23.62 %	Jeremy Allison, VA Linux
227	2.84 %	26.46 %	Simo Sorce, Xsec
203	2.54 %	28.99 %	Stefan Metzmacher, Unknown / Self-financed
185	2.31 %	31.31 %	Tim Potter, VA Linux
156	1.95 %	33.25 %	Christopher R. Hertel, Unknown / Self-financed
131	1.64 %	34.89 %	Jelmer Vernooij, Unknown / Self-financed
127	1.59 %	36.48 %	Steve Langasek, Unknown / Self-financed
125	1.56 %	38.04 %	Andrew Esh, TriCord ?
110	1.37 %	39.41 %	David Collier-Brown, SUN Microsystems
100	1.25 %	40.66 %	Volker Lendecke, Service Network GmbH
87	1.09 %	41.75 %	Rafal Szczesniak, Unknown / Self-financed
85	1.06 %	42.81 %	Jim McDonough, IBM
85	1.06 %	43.87 %	Alexander Bokovoy, Sam-Solutions
75	0.94 %	44.81 %	Luke Kenneth Casson Leighton, Unknown / Self-financed
71	0.89 %	45.70 %	Bradley W. Langhorst, University of New Hampshire
67	0.84 %	46.53 %	Don McCall, HP
62	0.77 %	47.31 %	Urban Widmark, Enlight
62	0.77 %	48.08 %	Ulf Bertilsson, Unknown / Self-financed
61	0.76 %	48.84 %	Mike Gerdts, Alcatel
59	0.74 %	49.58 %	Luke Howard, PADL Software Pty Ltd
57	0.71 %	50.29 %	David Lee, University of Durham
55	0.69 %	50.98 %	Juergen Hasch, Unknown / Self-financed
54	0.67 %	51.66 %	Steven French, IBM
51	0.64 %	52.29 %	Andreas Moroder, Unknown / Self-financed

Table A.7: A table listing the most active persons on the samba-technical mailinglist during 2002.

Main financial contributors during year 2002

# Mails	*Percentage*	*Aggregated*	*Company*
1583	37.82 %	37.82 %	Unknown / Self-financed
1014	24.22 %	62.04 %	VA Linux
426	10.18 %	72.22 %	Pansas
227	5.42 %	77.64 %	Xsec
139	3.32 %	80.96 %	IBM
125	2.99 %	83.95 %	TriCord ?
110	2.63 %	86.57 %	SUN Microsystems
100	2.39 %	88.96 %	Service Network GmbH
85	2.03 %	90.99 %	Sam-Solutions
71	1.70 %	92.69 %	University of New Hampshire
67	1.60 %	94.29 %	HP
62	1.48 %	95.77 %	Enlight
61	1.46 %	97.23 %	Alcatel
59	1.41 %	98.64 %	PADL Software Pty Ltd
57	1.36 %	100.00 %	University of Durham

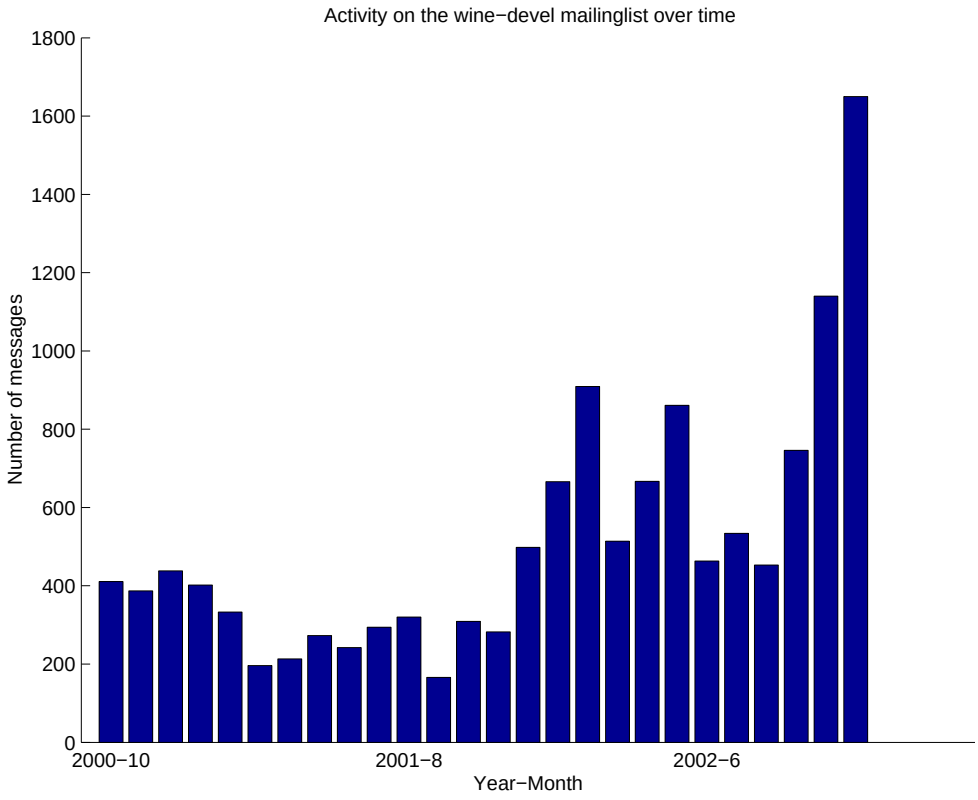
Table A.8: A table listing the most active companies on the samba-technical mailinglist during 2002. (Only persons in the previous figure is included.)

WINE

Background

WINE is an implementation of many of the APIs used by Microsoft Windows. The software enables Linux/Unix users to run Windows programs in their local environment. WINE is also a library, enabling companies to cross-compile their windows software to run under Linux/Unix without any performance loss. Several companies has been founded in order to take advantage of the possibilities given by the WINE project, bringing Microsoft Office to the Linux Desktop (Without the permission from Microsoft) as well as porting many applications and games. The increase in commercial interest made the developers change to a more restrictive license, forcing the companies to give back their work to the community. This has led to a fork of the project as the companies involved claimed that they were frightened of violating the Digital Millennium Copyright Act (DMCA). This study covers 14 034 messages from the wine-devel mailing-list between 2000-10-04 and 2002-12-17.

Activity over time



Main development contributors during year 2002

# Mails	*Percentage*	*Aggregated*	*Person*
768	8.28 %	8.28 %	Dimitrie O. Paun, Unknown / Self-financed
682	7.36 %	15.64 %	Alexandre Julliard, Codeweavers
364	3.93 %	19.57 %	Francois Gouget, Codeweavers
309	3.33 %	22.90 %	Eric Pouech, Unknown / Self-financed
293	3.16 %	26.06 %	Andreas Mohr, Unknown / Self-financed
278	3.00 %	29.06 %	Sylvain Petreolle, Unknown / Self-financed
265	2.86 %	31.92 %	Andriy Palamarchuk, Unknown / Self-financed
262	2.83 %	34.74 %	Steven Edwards, Unknown / Self-financed
228	2.46 %	37.20 %	Martin Wilck, Fujitsu-Siemens Computers
220	2.37 %	39.58 %	Patrik Stridvall, Unknown / Self-financed
209	2.25 %	41.83 %	Uwe Bonnes, Unknown / Self-financed
166	1.79 %	43.62 %	Dmitry Timoshkov, Codeweavers
166	1.79 %	45.41 %	Greg Turner, Unknown / Self-financed
165	1.78 %	47.19 %	Shachar Shemesh, Unknown / Self-financed
160	1.73 %	48.92 %	Dustin Navea, Unknown / Self-financed
156	1.68 %	50.60 %	Tony Lambregts, Unknown / Self-financed
143	1.54 %	52.14 %	Ove Kaaven, Transgaming
137	1.48 %	53.62 %	Bill Medland, ACCPAC International Inc.
122	1.32 %	54.93 %	Michael Cardenas, Lindows
106	1.14 %	56.08 %	Lionel Ulmer, Unknown / Self-financed
100	1.08 %	57.16 %	Duane Clark, Unknown / Self-financed
96	1.04 %	58.19 %	Vincent Beron, Unknown / Self-financed
95	1.02 %	59.22 %	Marcus Meissner, SUSE
82	0.88 %	60.10 %	Brett Glass, Unknown / Self-financed
80	0.86 %	60.96 %	Lawson Whitney, Unknown / Self-financed
75	0.81 %	61.77 %	Michael Stefaniuc, Unknown / Self-financed
75	0.81 %	62.58 %	Dan Kegel, Unknown / Self-financed

Table A.9: A table listing the most active persons on the wine-devel mailinglist during 2002.

Main financial contributors during year 2002

# Mails	*Percentage*	*Aggregated*	*Company*
3865	66.61 %	66.61 %	Unknown / Self-financed
1212	20.89 %	87.50 %	Codeweavers
228	3.93 %	91.43 %	Fujitsu-Siemens Computers
143	2.46 %	93.90 %	Transgaming
137	2.36 %	96.26 %	ACCPAC International Inc.
122	2.10 %	98.36 %	Lindows
95	1.64 %	100.00 %	SUSE

Table A.10: A table listing the most active companies on the wine-devel mailinglist during 2002. (Only persons in the previous figure is included.)

Conclusions

This paper is primarily a collection of data in order for me, and possible others, to prove various points in other papers. The separation of the data and my thoughts about it is highly intentional as this gives me the ability to verify my data from the parties involved without having my conclusions being questioned. This is not a statement saying that I do not want to have my conclusions discussed, simply that I want them to be discussed separately from this somewhat objective data, so that this data will remain unpolluted.

Further research

This paper reflects over a simple snapshot of a few Open Source projects (even though the snapshot is an aggregation during 2002) and there are therefore many aspects still to be investigated. How did this commercialization happened - it would be interesting to follow the community over time (forward or backward) to see how this all happened, and investigate if developers put in more effort now when they are paid than when they weren't.

It would also be interesting to further explore the thoughts of the minority that is still unpaid, even though they are doing as much work as they who are. Why do they keep on doing it, working for the community, when the community has been bough and replaced by corporations?

Any person interested in investigating this further in any way is welcome to contact the author in order to obtain useful tools as well as a database containing the archives.

On a side-note one should be able to observe a shift in activity over time as a result of the change in motivations. In the early stages, development efforts probably increased heavily during holidays, as the hobby-developers had more time then. When large parts of the efforts are done in corporations, development decreases when people leave the office. I am also convinced, even though this have not been studied here, that the working-hours also are affected from evenings and weekends to regular business hours.

Appendix B

A study of the Open Source projects that never took off

Introduction and method

This is a study of over 34 000 Open Source projects residing on the popular project repository SourceForge.net. The aim with this paper is to study small Open Source Project and show that there are large efforts done in projects that never become known to a wide audience.

The data used in this study were collected between 2002-12-17 and 2002-12-25. The data is essentially information about all the projects hosted on SourceForge.net. The reason for using SourceForge as the only primary data source is that they host a large number of projects and they do it in a very structured way, simplifying an automatic classification of projects. SourceForge.net have not taken any active part in this research, the data is collected by the author, from the website, and has not been verified by staff from SourceForge.net.

The observant reader will notice that there is quite a difference between the 34 262 projects (and 36 337 developers) covered in this paper and the number stated on www.sourceforge.net. This study only covers project that have been fully registered on the site and have an assigned administrator as well as some assigned property, we have thus filtered out projects that never started. We have also limited the research to developers actually assigned to at least one project, filtering out people registering on the site for other reasons.

This document is a collection of data, and will solely concentrate on the numbers, rather on what conclusions you might draw from it. There are two main reasons for this, the first one is that I would like to separate any discussion over the actual numbers from any conclusions made as a result of them and other data, and vice versa. Secondly, I believe in the mentality to release early and release often. If this study could be useful for other parties, I should not pollute it with my thoughts on other subjects,.

Concurrent Versions System

Most terms in this paper will not be explained further by me. There is however one concept to which I will return to several times, that is necessary to know in order to grasp this document. The Concurrent Versions System, or more commonly cvs is a piece of software used by most development projects in order to keep control of the versioning of the code as it evolves. In practice this means that each developer is able to keep a local copy of the code-base, develop on it and send back the changes in a formalized way. The system keeps track of possible conflicts (multiple persons trying to change the same piece of code) and enables the developer to check what has changed between different versions of the code.

Each project on SourceForge.net gets a own CVS-server for the developers to store their code, as this is the most convenient way to develop software, most projects use it and therefore it is a great source for measuring activity in a project. Each change is stored in the server and we can therefore go back and find out when development began, and possibly stalled.

Acknowledgments

I'd like to thank, not only the developers which made the research meaningful but also the folks at SourceForge.net for the data as well as the creators and teams behind some interesting software such as the Perl language, the MySQL database engine and the LyX editor for LaTeX. A special thank should also go to the people behind the LWP Perl-modules which saved me some work in collecting the data.

Results

Number of developers per projects

By counting the number and registered developers in each project we found that most project were in fact very small, over half of the projects were projects carried out by a single developer. This doesn't rule out that projects are developed in a collective fashion and there are possibilities for users to contribute without being listed as a developer. The core team, however tend to be very small for this kind of projects.

Projects	*Percentage*	*Aggregated*	*Developers*
20242	59.08 %	59.08 %	1
6223	18.16 %	77.24 %	2
2797	8.16 %	85.41 %	3
1515	4.42 %	89.83 %	4
952	2.78 %	92.61 %	5
575	1.68 %	94.29 %	6
444	1.30 %	95.58 %	7
1244	3.63 %	99.21 %	Other

Table B.1: A table showing how many developers the projects have. Projects with more than 7 developers have been grouped as Other.

Number of projects per developer

Even though the projects are small, there is a large specialization, most developer is only active in their own project.

Developers	*Percentage*	*Aggregated*	*Projects*
29608	81.48 %	81.48 %	1
4725	13.00 %	94.48 %	2
1280	3.52 %	98.01 %	3
392	1.08 %	99.09 %	4
323	0.89 %	99.98 %	Other

Table B.2: A table showing how many projects the developers are active in.

Intended Audience

Only a third of the projects are intended to be used by end-users, i.e. users without any knowledge about software development. Most of the tools developed are intended for developers and system administrators.

Audience	*Percentage*	*Aggregated*	*Projects*
20755	38.08 %	38.08 %	Developers
18044	33.11 %	71.19 %	End Users/Desktop
8833	16.21 %	87.40 %	System Administrators
4488	8.24 %	95.64 %	Other Audience
762	1.40 %	97.04 %	Information Technology
558	1.02 %	98.06 %	Education
1057	1.94 %	100.00 %	Other

Table B.3: A table showing who the projects are targeting. Items below 1% has been grouped as Other

Programming languages used

Projects	*Percentage*	*Aggregated*	*Programming Language*
9180	20.62 %	20.62 %	C
8690	19.52 %	40.13 %	C++
6998	15.72 %	55.85 %	Java
5278	11.85 %	67.71 %	PHP
3903	8.77 %	76.47 %	Perl
1976	4.44 %	80.91 %	Python
1065	2.39 %	83.30 %	Visual Basic
952	2.14 %	85.44 %	Assembly
948	2.13 %	87.57 %	Unix Shell
933	2.10 %	89.66 %	JavaScript
850	1.91 %	91.57 %	Delphi/Kylix
606	1.36 %	92.93 %	PL/SQL
591	1.33 %	94.26 %	Tcl
487	1.09 %	95.36 %	C#
2068	4.64 %	100.00 %	Other

Table B.4: A table showing how many developers the projects have. Items below 1% has been grouped as Other

Natural languages used

Even though English is the most popular natural language used in open source communities there are diversities as about a fifth of all projects utilizes other languages. In order for the community to be totally global, the participants must be able to communicate in a convenient way and English is the de facto standard on the Internet even though there might be other languages which are more represented within the actual project.

Natural Language	*Percentage*	*Aggregated*	*Projects*
14870	79.45 %	79.45 %	English
1533	8.19 %	87.65 %	German
911	4.87 %	92.51 %	French
593	3.17 %	95.68 %	Spanish
252	1.35 %	97.03 %	Russian
556	2.97 %	100.00 %	Other

Table B.5: A table showing how many developers the projects have. Items below 1% has been grouped as Other

Software license used

In order for open software to be open, the issues of copyright must be resolved. Large projects tend to create their own licenses such as the Sun Community License, and the Mozilla Public License in order to make them fit better to serve the interest of the companies. Small projects settle with the most commonly accepted licenses in order to protect their rights. The GNU Public License, drafted by the Free Software foundation seems to be overwhelmingly popular. This is the same license used in for instance the Linux Kernel and are considered to be viral as it forces derivate works to inherit this license.

License	*Percentage*	*Aggregated*	*Projects*
23484	68.16 %	68.16 %	GNU General Public License (GPL)
3390	9.84 %	78.00 %	GNU Library or Lesser General Public License (LGPL)
2287	6.64 %	84.64 %	BSD License
999	2.90 %	87.54 %	Public Domain
874	2.54 %	90.07 %	Artistic License
639	1.85 %	91.93 %	Other/Proprietary License
541	1.57 %	93.50 %	Apache Software License
524	1.52 %	95.02 %	MIT License
1716	4.98 %	100.00 %	Other

Table B.6: A table showing how many developers the projects have. Items below 1% has been grouped as Other

Target Environment

Environment	*Percentage*	*Aggregated*	*Projects*
4882	22.57 %	22.57 %	Console (Text Based)
4721	21.83 %	44.40 %	Web Environment
3415	15.79 %	60.19 %	Win32 (MS Windows)
3085	14.26 %	74.45 %	X11 Applications
2089	9.66 %	84.11 %	Other Environment
1431	6.62 %	90.73 %	No Input/Output (Daemon)
773	3.57 %	94.30 %	Gnome
508	2.35 %	96.65 %	KDE
344	1.59 %	98.24 %	Curses
224	1.04 %	99.27 %	Cocoa (MacOS X)
157	0.73 %	100.00 %	Other

Table B.7: A table showing how many developers the projects have. Items below 1% has been grouped as Other

Project Maturity

Development status	*Percentage*	*Aggregated*	*Projects*
10221	26.96 %	26.96 %	1 - Planning
7750	20.44 %	47.41 %	4 - Beta
6964	18.37 %	65.78 %	2 - Pre-Alpha
6369	16.80 %	82.58 %	3 - Alpha
5920	15.62 %	98.20 %	5 - Production/Stable
620	1.64 %	99.83 %	6 - Mature
63	0.17 %	100.00 %	Other

Table B.8: A table showing how many developers the projects have. Items below 1% has been grouped as Other

Target Operating System

Operating System	*Percentage*	*Aggregated*	*Projects*
13288	27.94 %	27.94 %	Linux
12181	25.62 %	53.56 %	OS Independent
4471	9.40 %	62.96 %	Windows 95/98/2000
3624	7.62 %	70.59 %	POSIX
3405	7.16 %	77.75 %	Windows
2551	5.36 %	83.11 %	Windows NT/2000
1148	2.41 %	85.53 %	SunOS/Solaris
852	1.79 %	87.32 %	MacOS X
825	1.73 %	89.05 %	FreeBSD
676	1.42 %	90.47 %	BSD
664	1.40 %	91.87 %	Other OS
594	1.25 %	93.12 %	Microsoft
3272	6.88 %	100.00 %	Other

Table B.9: A table showing which Operating Systems projects are targeting. Items below 1% has been grouped as Other

Type of software

Percentage	*Aggregated*	*Projects*	*Topic*
3388	6.03 %	6.03 %	Software Development
2667	4.75 %	10.78 %	Dynamic Content
1751	3.12 %	13.90 %	Site Management
1425	2.54 %	16.44 %	Internet
1374	2.45 %	18.89 %	Games/Entertainment
1139	2.03 %	20.92 %	Systems Administration
1084	1.93 %	22.85 %	Communications
1073	1.91 %	24.76 %	Role-Playing
1060	1.89 %	26.65 %	Front-Ends
1048	1.87 %	28.51 %	Database
1031	1.84 %	30.35 %	Other/Nonlisted Topic
969	1.73 %	32.07 %	WWW/HTTP
883	1.57 %	33.65 %	Networking
842	1.50 %	35.15 %	Education
829	1.48 %	36.62 %	Build Tools
799	1.42 %	38.05 %	Chat
762	1.36 %	39.40 %	Code Generators
755	1.34 %	40.75 %	Security
745	1.33 %	42.07 %	Office/Business
743	1.32 %	43.40 %	Internet Relay Chat
713	1.27 %	44.67 %	CGI Tools/Libraries
694	1.24 %	45.90 %	Scientific/Engineering
653	1.16 %	47.07 %	File Sharing
651	1.16 %	48.23 %	Artificial Intelligence
649	1.16 %	49.38 %	Desktop Environment
638	1.14 %	50.52 %	Multi-User Dungeons (MUD)
630	1.12 %	51.64 %	Monitoring
606	1.08 %	52.72 %	Simulation
596	1.06 %	53.78 %	Interpreters
25952	46.22 %	100.00 %	Other

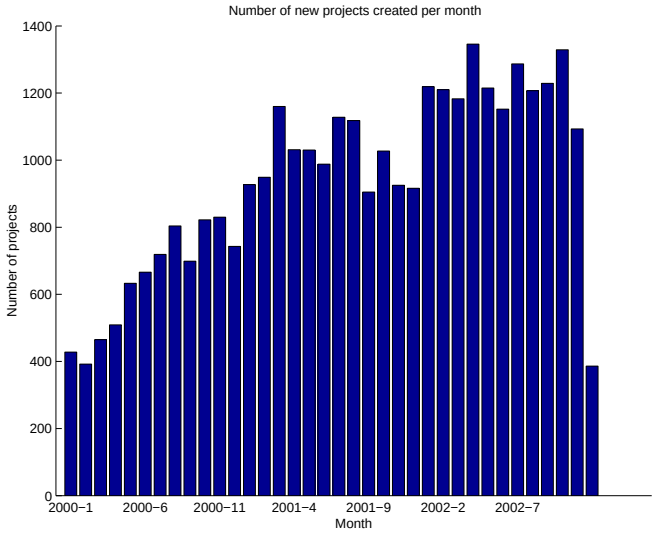
Table B.10: A table showing how many developers the projects have. Items below 1% has been grouped as Other

New projects over time

By aggregating the created-date of each project I were able to reconstruct the growth of projects during the last few years.

Month	*Projects*
428	2000-1
392	2000-2
465	2000-3
509	2000-4
633	2000-5
666	2000-6
719	2000-7
804	2000-8
699	2000-9
822	2000-10
830	2000-11
743	2000-12
927	2001-1
949	2001-2
1160	2001-3
1031	2001-4
1030	2001-5
988	2001-6
1128	2001-7
1118	2001-8
905	2001-9
1027	2001-10
925	2001-11
916	2001-12
1219	2002-1
1210	2002-2
1183	2002-3
1346	2002-4
1215	2002-5
1152	2002-6
1287	2002-7
1207	2002-8
1229	2002-9
1329	2002-10
1093	2002-11
386	Other

Table B.11: A table showing how many new projects that were created each month.

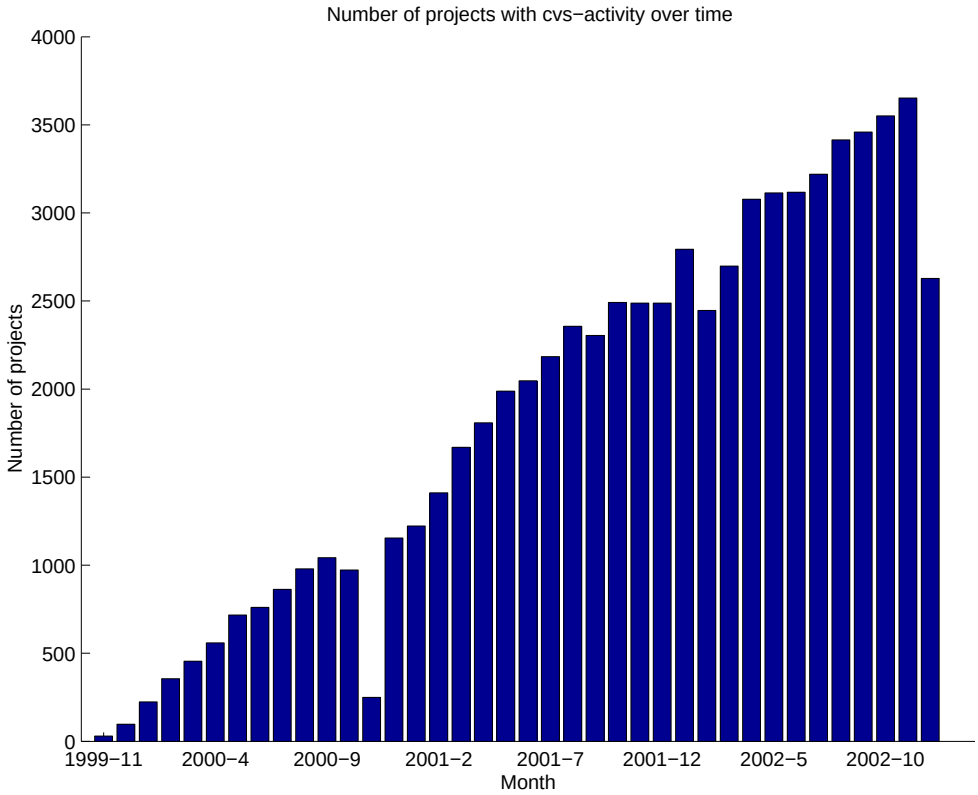

Number of new projects created per month
Number of projects
Month
0
200
400
600
800
1000
1200
1400
2000−1
2000−6
2000−11
2001−4
2001−9
2002−2
2002−7

Cvs-activity over time

This data shows how many projects which had cvs-activity (i.e. changes in the source code) each month.

Month	*Projects*
30	1999-11
97	1999-12
224	2000-1
355	2000-2
455	2000-3
559	2000-4
717	2000-5
761	2000-6
863	2000-7
979	2000-8
1043	2000-9
973	2000-10
249	2000-11
1154	2000-12
1222	2001-1
1411	2001-2
1669	2001-3
1808	2001-4
1988	2001-5
2047	2001-6
2184	2001-7
2356	2001-8
2304	2001-9
2492	2001-10
2488	2001-11
2488	2001-12
2794	2002-1
2446	2002-2
2698	2002-3
3078	2002-4
3114	2002-5
3117	2002-6
3220	2002-7
3415	2002-8
3459	2002-9
3551	2002-10
3652	2002-11
2628	2002-12

Table B.12: A table showing how many projects that had cvs-activity each month.

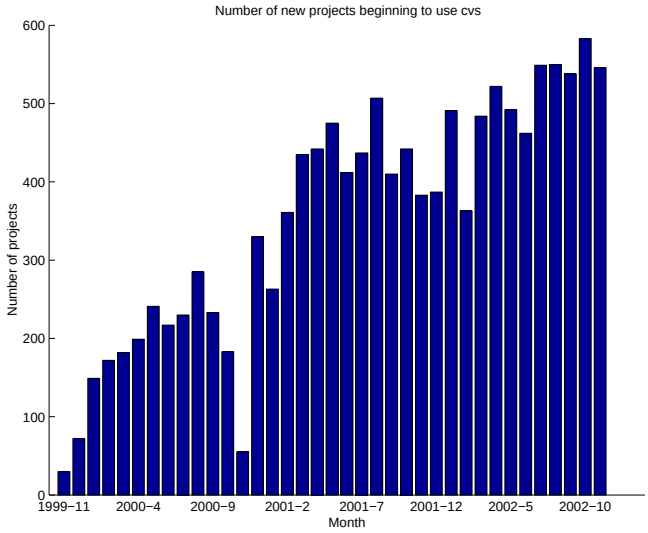

Number of projects with cvs−activity over time
Number of projects
4000
3500
3000
2500
2000
1500
1000
500
0
1999−11
2000−4
2000−9
2001−2
2001−7
2001−12
2002−5
2002−10
Month

Initial cvs-activity for each project

This is an alternative way to measure when projects were created. This data shows how many projects which had their first activity on the cvs-server each month.

Month	*Projects*
361	2001-2
435	2001-3
442	2001-4
475	2001-5
412	2001-6
437	2001-7
507	2001-8
410	2001-9
442	2001-10
383	2001-11
387	2001-12
491	2002-1
363	2002-2
484	2002-3
522	2002-4
492	2002-5
462	2002-6
549	2002-7
550	2002-8
538	2002-9
583	2002-10
546	2002-11
2841	Other

Table B.13: A table showing how many projects began to use cvs each month.

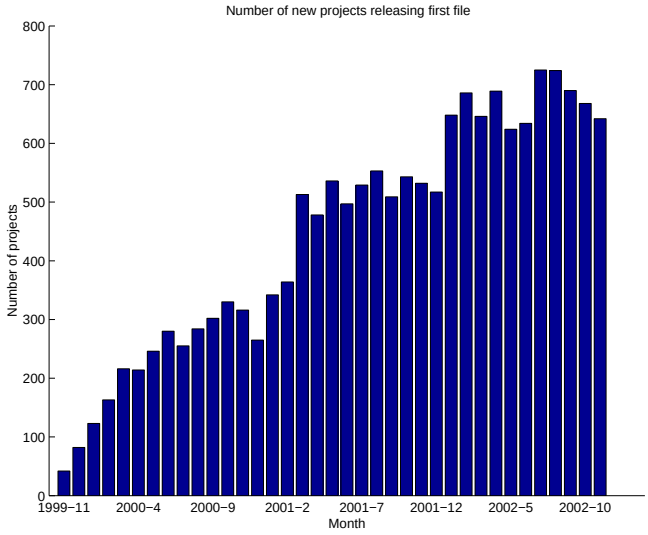


First file downloaded

After some development, eventually there should be a first version of the software. This data shows how many projects which released their first file each month.

Month	*Projects*
42	1999-11
82	1999-12
123	2000-1
163	2000-2
216	2000-3
214	2000-4
246	2000-5
280	2000-6
255	2000-7
284	2000-8
302	2000-9
330	2000-10
316	2000-11
265	2000-12
342	2001-1
364	2001-2
513	2001-3
478	2001-4
536	2001-5
497	2001-6
529	2001-7
553	2001-8
509	2001-9
543	2001-10
532	2001-11
517	2001-12
648	2002-1
686	2002-2
646	2002-3
689	2002-4
624	2002-5
634	2002-6
725	2002-7
724	2002-8
690	2002-9
668	2002-10
642	2002-11

Table B.14: A table showing how many new projects that released it’s first file each month.

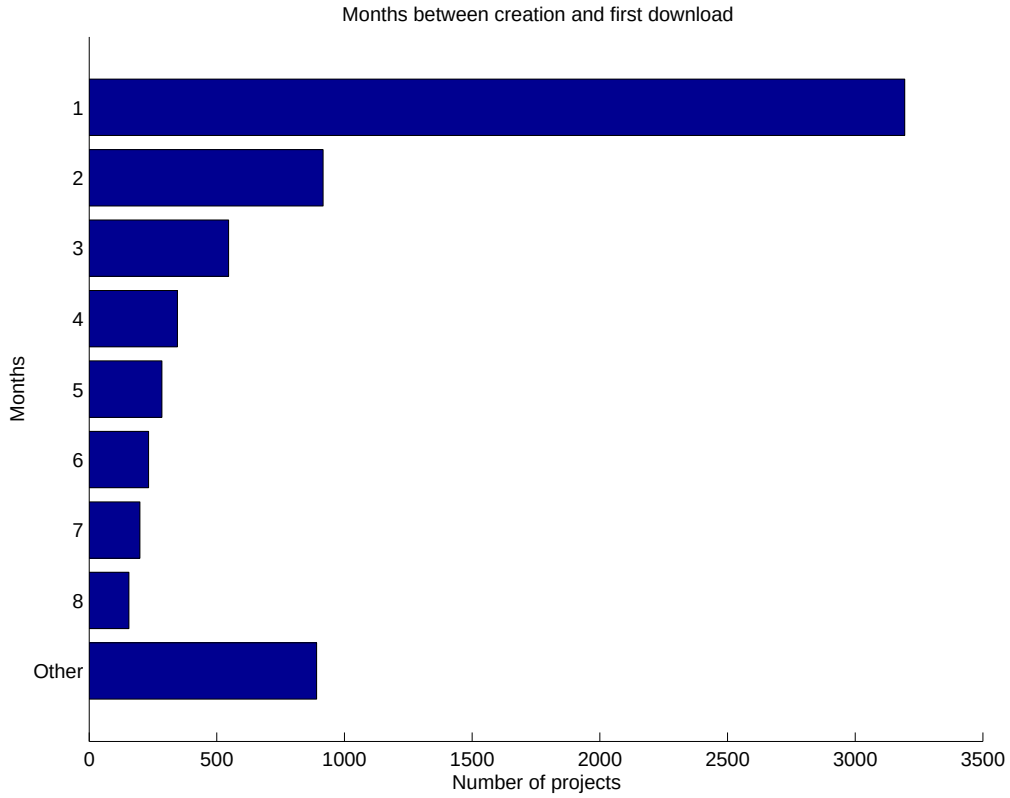

Number of new projects releasing first file
Number of projects
0
100
200
300
400
500
600
700
800
1999−11
2000−4
2000−9
2001−2
2001−7
2001−12
2002−5
2002−10
Month

Time before cvs-activity began

How long is the path between though and action, this data shows how long time that elapsed between the project was created and the first lines of code were added to the server.

Months Percentage	*Aggregated*	*Projects*	
3194	24.07 %	24.07 %	1
916	6.90 %	30.98 %	2
546	4.12 %	35.09 %	3
346	2.61 %	37.70 %	4
285	2.15 %	39.85 %	5
233	1.76 %	41.61 %	6
199	1.50 %	43.11 %	7
156	1.18 %	44.28 %	8
891	6.72 %	51.00 %	Other

Table B.15: A table showing the number of months between the creation of a project and its first download.

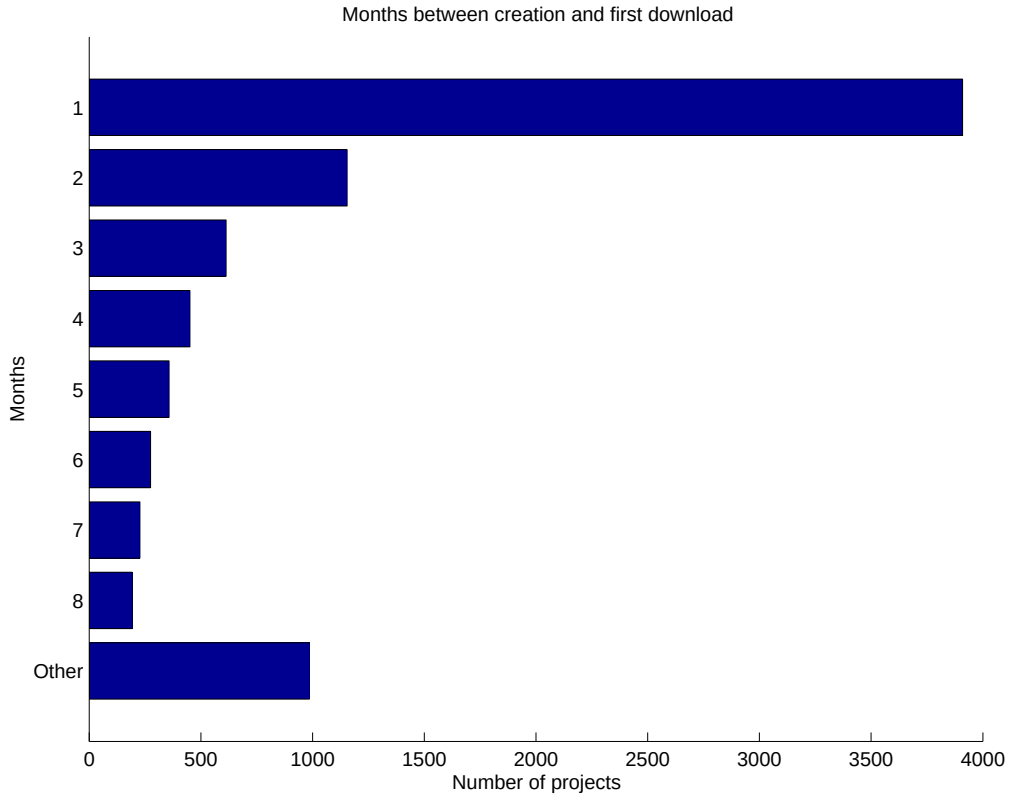


Time before first file downloaded

This data shows how many months that elapsed between the creation of the project and the first file release.

Months	*Projects*
3909	1
1155	2
613	3
451	4
358	5
275	6
227	7
193	8
986	Other

Table B.16: A table showing the number of months between the creation of a project and its first download (only projects with releases counted).

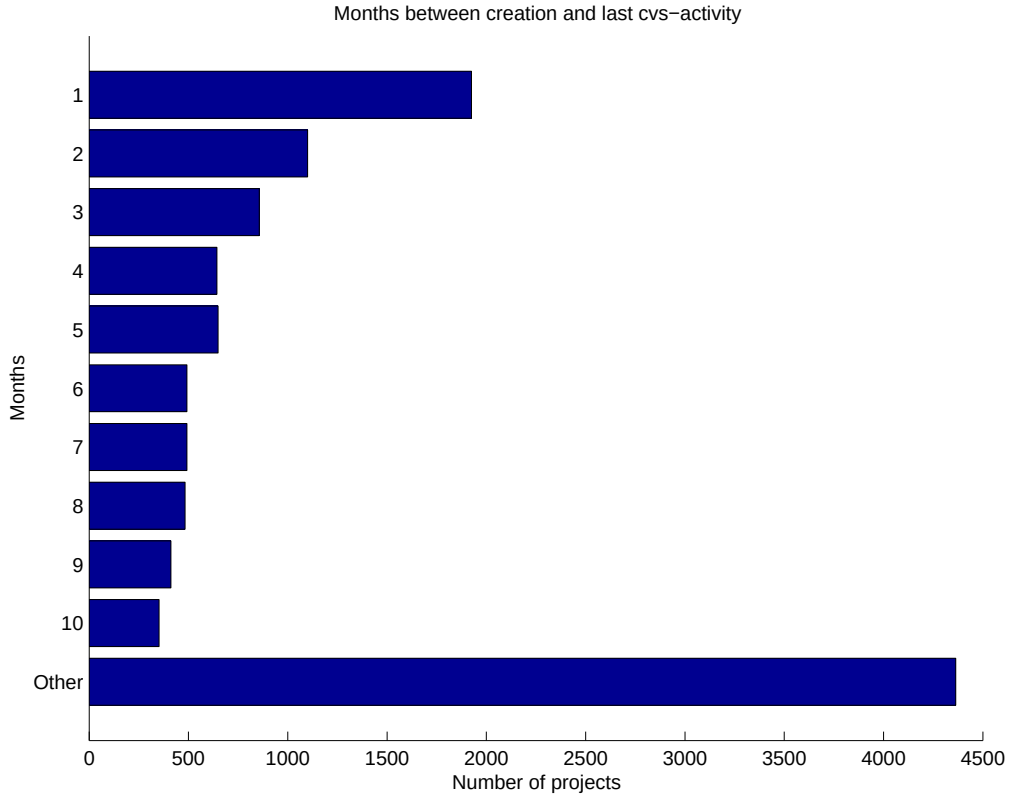


Time until cvs-activity stalled

This data shows how many months elapsed between the first cvs-activity and the last. i.e. how long the project were active before it died.

Months	*Percentage*	*Aggregated*	*Projects*
1925	5.62 %	5.62 %	1
1100	3.21 %	8.83 %	2
857	2.50 %	11.33 %	3
643	1.88 %	13.21 %	4
649	1.89 %	15.10 %	5
492	1.44 %	16.54 %	6
492	1.44 %	17.97 %	7
483	1.41 %	19.38 %	8
412	1.20 %	20.59 %	9
352	1.03 %	21.61 %	10
4364	12.74 %	34.35 %	Other

Table B.17: A table showing the number of months between the creation of a project and its last cvs-activity. (only dead projects included)



Maturity of stalled projects

This data shows the level of maturity reached by the projects that have stalled.

Month	*Percentage*	*Aggregated*	*Projects*
7259	34.62 %	34.62 %	1 - Planning
4735	22.58 %	57.20 %	2 - Pre-Alpha
3241	15.46 %	72.65 %	3 - Alpha
3057	14.58 %	87.23 %	4 - Beta
2365	11.28 %	98.51 %	5 - Production/Stable
269	1.28 %	99.79 %	6 - Mature
43	0.21 %	100.00 %	7 - Inactive

Table B.18: A table showing the maturity of the projects which are inactive.

Number of active versus inactive projects

Active	*Inactive*	*Percentage*	*Month*
30	96	31.25 %	1999-11
97	224	43.30 %	1999-12
224	477	46.96 %	2000-1
355	702	50.57 %	2000-2
455	967	47.05 %	2000-3
559	1205	46.39 %	2000-4
717	1500	47.80 %	2000-5
761	1761	43.21 %	2000-6
863	2037	42.37 %	2000-7
979	2363	41.43 %	2000-8
1043	2640	39.51 %	2000-9
973	2944	33.05 %	2000-10
249	3251	7.66 %	2000-11
1154	3512	32.86 %	2000-12
1222	3835	31.86 %	2001-1
1411	4210	33.52 %	2001-2
1669	4688	35.60 %	2001-3
1808	5126	35.27 %	2001-4
1988	5575	35.66 %	2001-5
2047	5985	34.20 %	2001-6
2184	6453	33.84 %	2001-7
2356	6945	33.92 %	2001-8
2304	7323	31.46 %	2001-9
2492	7743	32.18 %	2001-10
2488	8118	30.65 %	2001-11
2488	8457	29.42 %	2001-12
2794	8895	31.41 %	2002-1
2446	9328	26.22 %	2002-2
2698	9764	27.63 %	2002-3
3078	10202	30.17 %	2002-4
3114	10640	29.27 %	2002-5
3117	11059	28.19 %	2002-6
3220	11545	27.89 %	2002-7
3415	12004	28.45 %	2002-8
3459	12476	27.73 %	2002-9
3551	12922	27.48 %	2002-10
3652	13240	27.58 %	2002-11
2628	13262	19.82 %	2002-12

Table B.19: A table showing how many projects that were active/inactive each month (only projects using cvs included).