Uniwersytet Łódzki

Wydział Ekonomiczno - Socjologiczny

Kierunek Informatyka i Ekonometria
Specjalność Informatyka Ekonomiczna

Błażej Borucki
Nr albumu 109432/s

The Economical Aspects of Free Software and Open Source Software Solutions in Modern Business

Praca magisterska napisana
pod kierunkiem dr. Konrada Woźniackiego

Łódź 2006

1 Table of contents

2 The general overview of my master thesis

This thesis investigates topic of advantages of Free and Open Source Software (FOSS) solutions from the commercial users' and developers' point of view . During the last couple of years FOSS community has transformed from the group of enthusiasts, who were often regarded as anarchists and utopians, into a group of developers and users wanted by every producer or software' vendor. This paper focuses on differences between development process of proprietary and non-proprietary software as well as superiority of FOSS solutions over proprietary software during usage period as: considering quality, ease of use, comfort of creation and usage of this type of software.

I have divided this desideratum into two parts: an introduction into economical point of view of FOSS (chapters 3-5) and an attempt to empirical description of Free Software development process (chapter 6).

First part of this thesis presents the history, some features and aspects of creating and using non-proprietary software and already mentioned differences. The aim of this chapter is to give an overview of Free/Open Source Software from the point of view of subject, which wants to optimise the utility of software (whether it is a tool or the end-product) along with continuous cost minimizing. In this part my opinions will be presented as well as conclusions and ideas derived from existing data, observations and researches.

Second part contains study with use of statistical data on projects hosted on sourceforge.net site. In chapter 6 I tried to prove that such factors like number of developers, number of downloads or traffic generated on message boards make difference between process of developing Free Software from developing proprietary solutions. Data for my research, was provided by faculty of Computer Science & Engineering of University of Notre Dame. In fact this chapter is an empirical picture of chapter 5.4, which describes differences in proprietary and non-proprietary software products life cycle.

3 An introduction to Free and Open Source Software

3.1 How the saga begun

Most of the "software consumers" are used to the fact, that they have to pay for every single piece of application installed on their computers and the only thing they get is program in its executable form. Not only the lack of source code restricts the software usage, also licenses include clauses that forbid particular ways in which application could be used. Such software - called proprietary software - may become very inoperable in many specific environments like scientific labs or colleges. The creator of non-proprietary software movement is Richard Stallman. He described himself[1] as inventor of EMACS[2] editor and MIT Artificial Intelligence Lab employee who have been working on, among other things, compilers, editors and debuggers.

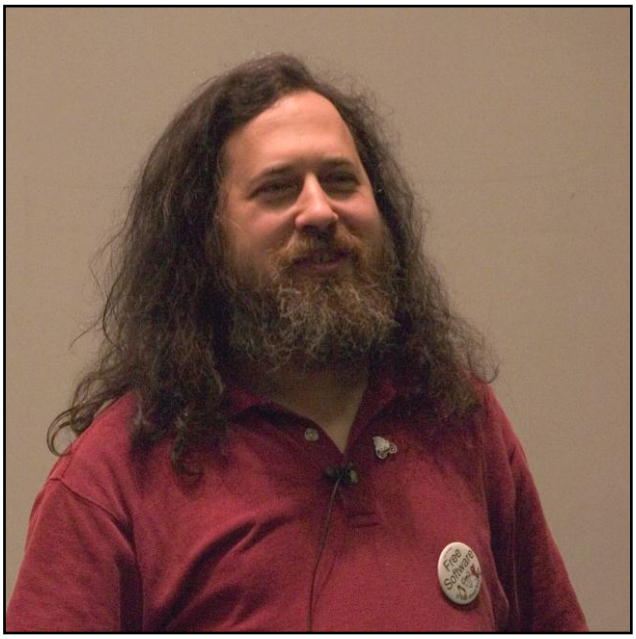

Illustration 1: Richard Stallman
Source: "The danger of software patents" - http://jimmysweblog.net/

In 1971, when Richard Stallman started his career most of software was proprietary, but during scientific researches and projects Stallman and his colleagues used non-proprietary software exclusively. It was a result of fact, that possibility of maintaining

1 Richard Stallman - The Initial Announcement ([27]).
2 EMACS is the extensible, customizable, self-documenting real-time display (text) editor.

and developing the application - given by source code accessibility - hastens projects, turns down its cost and increases application flexibility and feasibility. The problem encountered by Richard Stallman was, that almost all of software that were used by non-programmers was proprietary and owned not by users who bought it but by companies, which sold it giving them right to prevent users from customizing application to fit their needs. In the early 80's Stallman discovered that drivers for one of the printers, which he was trying to use, has some critical bugs. These drivers could not be corrected because of the license, which bans any changes in the code. He came to conclusion, that creating hardware drivers by numerous groups of its users, would be the much more effective. Furthermore, and at the end of the day drivers developed this way should be more reliable than ones supplied by hardware vendor. It is said that incident with non-free printer driver ultimately resulted in founding the GNU[3] Project (1984) and Free Software Foundation - FSF (1985).

3.1.1 Meaning and extraction of terms 'Free Software' and 'Open Source Software'

The definition of the term 'Free Software' has it's origin in genesis of GNU Project. The comparative definition maintained by Free Software Foundation indicates the proper way of understanding meaning of the word 'free'. "*Free software* is a matter of liberty, not price. To understand the concept, you should think of *free* as in *free speech*, not as in *free beer"*[4]. What needs to be emphasize is the importance of verity of meaning of word 'free'. Andrew Wheeler[5] found in 'A Linux Today' posting[6] very pertinent way to express meaning of the word free – "Free, as in free speech, free beer, free cocaine (*the first one is on me*)".

3 GNU is the recursive acronym for GNU is Not Unix.
4 Free Software Foundation - The Free Software Definition ([6]).
5 Andrew Wheller - www.dwheeler.com
6 [[30].

Illustration 2: GNU logo - version 1
Source: www.gnu.org

Illustration 3: GNU logo - version 2
Source: www.gnu.org

By FSF definition any program is free software if its users have four freedoms:

freedom 0: "The freedom to run the program, for any purpose ().
freedom 1: The freedom to study how the program works, and adapt it to your needs). Access to the source code is a precondition for this.
freedom 2: The freedom to redistribute copies so you can help your neighbour
freedom 3: The freedom to improve the program, and release your improvements to the public, so that the whole community benefits. Access to the source code is a precondition for this."

While referring to 'freedom of speech' one should not forget popular sentence from Marx brothers show "...freedom of speech means that you can shout *theatre* in crowded fire...". Each one of mentioned freedoms protects rights of author of original source code - user freedom to use, change, and redistribute code does not confine authors and other users' privileges and rights[7].

Another commonly used term is Open Source Software (OSS). Definition of OSS is maintained by Open Source Initiative (OSI) and states that open source does not mean just access to source code but distribution terms of open-source software must grant rights to redistribute (selling or giving away) software as component or derived work. It says that the changed source code (if license permit distribution of such) must be integral with original one, no persons, groups or fields of endeavour must be discriminated and the license must not be specific to a product and restrict other software. The term "Open Source Software" was used for the first time in 1998 by people who belonged to free software community. The term Open Source refers to

7 This matter will be discussed in chapter devoted to licensing.

development methodology; free software is a social movement and these two terms should not be used as synonyms. Unfortunately they are because of ambiguous meaning of 'free software'. FSF emphasize two facts - that FS doesn't mean "Software you can get for zero price" and that term "*Open Source* does not solve any problems, and in fact creates ones"[8]. On the other hand Open Source Initiative states that equivocal meaning of "free software" makes it misleading to the point where it becomes useless[9]. Each time when FSF says that 'free' is for 'freedom' OSI reminds of fact, that some software is called free because it costs no money. In OSI document "Why *Free* Software is too Ambiguous" public domain software and Microsoft Internet Explorer are mentioned as examples of software which is called 'free' (MsIE can be obtain for zero-price) but have nothing in common with FS defined by FSF. In mid-2004 web-content analysis on usage frequencies of the phrases 'open source' and 'free software' was made by the President of OSI. In technology trade press and among developers usage of term OSS exceed usage of FS by 90%, on general Web the usage ratio is 80% - 20%. Because of equivocal meaning of 'free' 80% of Web searches gave false-positive result. Such results may signify - how the OSI interpret them - that all efforts put by FSF to solve problem of ambiguous meaning of expression 'free software' were ineffectual. FSF defend itself stating, that it is better to use term ’free software’ because the 'freedom' is much more than ability to look at source code - which stands behind phrase 'Open Source Software' but agrees, that using term 'Free Software' makes some people uncomfortable. More effective way to present idea of FS is to show practical benefits rather than describe ethical issues, responsibilities and 'aspect of freedom'. According to Richard Stallman his decision to start GNU project was based on similar spirit that can be found in phrase attributed to Hillel:

“If I am not for myself, who will be for me?
If I am only for myself, what am I?
If not now, when?"[10]

Effect of such statements often infers arguments that free software social movement is rather ideological and because of fact that OSI focuses on technical matters OSS is associated with adaptable and commercial features of non-proprietary software. Reasons why such approach is completely vicious will be analysed in further part of this

8 Free Software Foundation - Why ”Free Software” is better than ”Open Source”’ ([7]).
9 Open Source Initiative - Why "Free" Software is too Ambiguous ([23]).
10 Open Sources ([5]) - part devoted to Richard Stallman.

chapter, dedicated to comparison of OSS and FS licenses. In the first place I am going to present Free Software Foundation and Open Source Initiative.

3.1.2 Free Software Foundation and Open Source Initiative presentation

On 27th of September 1983 Richard Stallman sent initial announcement[11] of GNU project to two newsgroups (net.unix-wizards and net.usoft) containing explanation why he must write GNU and what it will become. In Stallman's scheme GNU has been described as UNIX compatible system consisting of kernel and some utilities needed to write and run C programs, which will be given away free to anybody who can use it (there was no commentary on meaning of term 'free' in Stallman's post). His purpose was to be able to share any program he likes with other people who like it too and to use computers not being forced to use software, which cannot be preceded that way[12]. In FSF's overview of the GNU project[13] one can find note on why name "GNU" was chosen for this Free Unix project:

> *"The name GNU was chosen because it met a few requirements; firstly, it was a recursive acronym for GNU's Not Unix, secondly, because it was a real word, and thirdly, it was fun to say (or Sing)."*

Before FSF was founded by Stallman he made necessary steps to protect his future work from his current employer (MIT) claims. He quit his job at MIT AI lab in January 1984 because Institute could have claimed to employee's own work. Stallman begun writing non-proprietary software and he did not want it be used in an unintentional manner. Afterwards Stallman published The GNU Manifesto - a document amplifying initial announcement. Stallman has added the information on GNU, ways of contributing to project, benefits from project and refutation to objections to GNU goals[14]. To promote ideas mentioned in the initial announcement and manifesto in 1985 Free Software Foundation (FSF) was founded. Foundation encourages developing and uses free software as well as it causes a political and ethical freedom issues in software

11 Richard Stallman - The Initial Announcement ([27]).
12 Richard Stallman - The Initial Announcement ([27]).
13 Free Software Foundation - Overview of the GNU Project ([8]).
14 Richard Stallmann - The GNU manifesto ([28]).

usage to become more extensively known[15]. In 1998 from free software community new institution emerged - Open Source Initiative (OSI). Some people began using the term 'open source software' (OSS) instead of 'free software' to diversify their approach to non-proprietary software. The separation itself was a reaction to Netscape announcement about plans to make source code of Netscape Communicator available for free licensing purposes[16]. The term 'open source' came out on February 3rd 1998 when the session on new 'free software' strategy was held. According to OSI papers representatives of 'Foresight Institute' (Todd Anderson and Chris Peterson), 'Linux International' (John Hall and Larry Augustin) and 'Silicon Valley Linux User's Group' (Sam Ockman) were presented among others.

They decided to abandon attitude of FSF and make their new viewpoint more pragmatic and focused on business aspects of developing and maintaining non-proprietary software. The OSI become alternative to FSF on non-proprietary software market, even FSF competitor giving developers and users possibility to use non-proprietary software

Illustration 4: The Free Software Foundation Logo
Source: www.gnu.org

Illustration 5: The Open Source Initiative Logo
Source: www.opensource.org

without freedom.

3.2 *Comparison of Free Software and Open Source Software Licenses*

3.2.1 Introduction

As I have already mentioned, term 'Open Source Software' is frequently used as 'Free Software' alternative, which causes the misinterpretation of the philosophy and goals of OSS and FS contributors. In practice, most of software which fill requirements of one

15 Free Software Foundation description([9]).
16 Netscape Announces Plans To Make Next-Generation Communicator Source Code A Viable.... – ([20]).

definition also meets the other one but 'open source' not always gives users the freedom to do anything they want with the code (as long as what they need to do is conformable to the piece of software license). Licenses, which cover Free Software, Open Source Software and proprietary software shows approach and purpose of developers. In reality, the majority of end users rather ask about reliability, performance, ease of use and other technical aspects of system/application software than what determined its authors to prefer one license to others. That is how Richard Stallman explains differences between FS and OSS:

"Free software and open source are the slogans of two different movements with different philosophies. In the free software movement, our goal is to be free to share and cooperate. We say that non-free software is antisocial because it tramples the users' freedom, and we develop free software to escape from that. The open source movement promotes what they consider a technically superior development model that usually gives technically superior results. The values they cite are the same ones Microsoft appeals to: narrowly practical values."

3.2.2 Licenses

A license in general - found on Collaborative International Dictionary of English[17] - is:

> ***License: an authority or liberty given to do or forbear any act, especially, a formal permission from the proper authorities to perform certain acts [...] which without such permission would be illegal.***
>
> ***Collaborative International Dictionary of English***

The licenses, which I am going to analyse, mean the terms and conditions for use, reproduction and distribution of software and/or source code. A typical license is composed of several parts: its proprietor - in most cases it is a company, institution person (group of people), introduction or preamble - this part usually describe contents of license in general and defines terms used in license body. The most important sections are those which cover rights, obligations, liabilities and warranty, which refer to authors and/or end users.

17 The Collaborative International Dictionary of English v.0.48

Free Software Foundation maintains and encourages to use GNU General Public License[18] (GNU GPL). The GNU GPL preamble articulates importance of freedom given to users of free Software. This license do not covers any another activities than copying, distributing and modifying the program - in GNU GPL term 'program' reefers to any program or its derivatives. As FSF states in GNU GPL FAQ that the crucial aspect of free software is a cooperation between users, that means sharing fixes and improvements with other users. To achieve this goal users are free to change the original source code or its derivatives. Changed program may be used privately even by commercial organizations without obligation to release new version. But if someone decide to share modified version the GPL requires such person to make the modified source code available to the public under GPL. To fulfil all GPL requirements modified files must carry notices stating who and when changed them. If the program is interactive it must display announcement including copyright notice and statement about warranty - if there is warranty provided by author or not. When the program is distributed under GPL the source-code must be given with executable version or the author must offer a complete machine-readable copy of the corresponding source code for a charge (which cannot be higher than cost of distribution). The GPL states that "any attempt [...] to copy, modify, sublicense or distribute the Program [under different license] is void, and will automatically terminate your rights under this License". In fact if user do not want to accept the license, do not have to do that and can still use the program, however such procedure causes in loosing rights to copy, modify or sublicense the program. Because user does not sign the license, these actions signify acceptance of it. There is another version of GNU GPL called GNU Lesser General Public License (GNU LGPL). The word 'lesser' states that users' rights are less protected by the license and developers are less competitive on software market. The LGPL is used for licensing libraries for two reasons:

- Because the program or source code released under LGPL can be utilize in proprietary software LGPL gives prospects to encourage larger group of people to use given solution and fix a new standard. That is why developers - and at the end of the day users - avail of Lesser GPL while releasing libraries.

18 Free Software Foundation - "GNU Public License" ([10]).

- More often LGPL is used when the 'Free'[19] library has the same application as widely used non-free library, in such situation it is aimless to limit library to free software only.

It may be a little bit dangerous to use LGPL for wider adaptation and the license preamble states to be careful with lesser version because of its disadvantages. Both – GPL and LGPL – include section about warranty, which states, that program released under *GPL is distributed with hope that will be useful but without any warranty – the program and its source code comes "AS IS". De facto 98% of proprietary software licenses do not states licensor liability but when source code is available to the public chance for abuse is much lower than in case of proprietary software. In general to make program a free software (and to make sure, that all works derived from such program will be free software too) method called "copyleft" is used. This term takes it origin from "copyright" which is used – as states FSF – to take away users freedom so when the copyrights guarantee freedom the opposite term has been adapted. The minor feature of copylefting is to grant rights "...to use, modify and redistribute the programs' code or any program derived from it but only if the distribution terms are unchanged"[20]. Another very important and widely used license is Open Software License (OSL) which

"*applies to any original work of authorship (the 'Original Work') whose owner ('the Licensor') has placed the following notice immediately following the copyright notice for the Original Work: Licensed under the Open Software License version 2.1*"[21].

The main difference between GPL and OSL is that OSL grants rights to prepare derivative works, copy and distribute both – the original and derived - program and/or source code to the public under OSL. It seems to give the very same freedom to users as GPL but in fact it does not. Where GPL makes certain that users can modify and redistribute modified program under the same license, the OSL leave the licensor admissibility to forbid some actions like the Trolltech's Q Public License (QTPL).

19 free as in free speech, not free beer :)
20 Free Software Foundation – www.gnu.org/licenses/licenses.html.
21 Open For Business – Philosophy ([22]).

Illustration 6: Qt logo
Source: www.trolltechh.com

Illustration 7: KDE logo
Source: www.kde.org

This Norwegian company grants rights to copy and distribute unmodified (the whole program package must remain unchanged) form of software code and to distribute ones modifications in form absolutely separated from the original program (like patches) under the QTPL. Only "initial developer" can distribute change versions of program with modifications made by any third party. Trolltech's programmers have created qt libraries – the C++ libraries for cross-platform gui programming, successfully utilized in K Desktop Environment project – which are fully commercial, proprietary product for Microsoft Windows and open source for other operating systems and to increase the quality of its products makes developing and maintaining processes more controlled. It is really efficient approach, but if such company – the "initial developer" - bankrupt or simply shuts down itself, program under license like QTPL could be taken over by creditors or be left without maintainers. To guarantee that the users and developers community contribution won't be lost, Trolltech and KDE developers team made an agreement. It states, that if Trolltechs won't be able to continue maintaining qt libraries project on the same conditions as now, libraries would become Free Software. The fully different approach is represented by the BSD License, which allows everybody to use original or modified source code for any purpose – even as a part of strictly proprietary software. Thanks to that, many good solutions can be generally used in different types of software, but BSD license caused fears of free-software community extinction in result of using free software in proprietary projects for wide range without rewarding its contributors. Because of that, BSD license is becoming less popular recently.

4 Why Free/ Open Source Software

Starting to write my thesis I asked this question: why customers should choose FOSS solutions for their home and business and why some developers contribute to such projects. Answer for the first part is much easier because of some factors, which can be easily measured or at least classified as desired or not.

4.1 Technical aspects

Features which are connected with Free Software, are not only ideologically fair from end user point of view. I will try to discuss some fundamental attributes of a computing system/ software design and its' implementation in Free/ Open source Software solutions. The RAS – Reliability, Availability and Serviceability are features regarded as ones, which describe software quality in the widest way.

> ***Availability: degree to which a system suffers degradation or interruption in its service to the customer as a consequence of failures of one or more of its parts.***
>
> ***The Free On-line Dictionary of Computing***

In case of proprietary software the knowledge on internal influences in application is unavailable, each and every proprietary application works like black box[22] due to the way it is distributed – binary version of program covered by license prohibiting at least all activities which may help in analysing program internal logic. The software package covered by GPL can be analysed and – if modifications are required – changed before used for strategically purposes. End users have possibility to report bugs or fix them. It is strictly connected with Serviceability, which is the ease with which corrective maintenance or preventative maintenance can be performed on a system. Higher serviceability improves availability and reduces costs of service and maintenance. Proving that Free and Open Source Software has higher serviceability factor is nearly needless because of the fact, that proprietary software can't be serviced by any third party users/ developers who would like to, due to lack of access to source code and proper technical documentation. More on maintaining FLOSS reader can find in

22 Black box - an abstraction of a device or system in which only its externally visible behaviour is considered and not its implementation or "inner workings" ('The Free On-line Dictionary of Computing').

chapters 4.2, 4.2.3 and 6. In this chapter I mention some tests and experiments I have found in David A. Wheeler writing.

4.1.1 Reliability

Reliability as third part of RAS is an attribute of any system that consistently produces the same results, preferably meeting or exceeding its specifications. If users put for example value of 10 into the black box and it returned 42 it is supposed to return 42 every time 10 is given as an argument *ceteris paribus* and this is the key to measure this issue.

> ***Reliability:. the ability of a system or component to perform its required functions under stated conditions for a specified period of time.***
>
> ***The Software Engineering Institute- Terms Glossary***

Many companies interested in FOSS made run tests and done some experiments to check whether or not Linux and other Free Software solutions are reliable. IBM run a series of extremely stressful tests for 30 and 60 days periods[23] and after these tests IBM claimed that tests demonstrated that "the Linux kernel and other core OS components are reliable and stable ... and can provide a robust, enterprise-level environment for customers over long periods of time". When Blur Research[24] had Windows NT and GNU/Linux running on relatively old machines for 12 months Windows NT crashed 68 times and GNU/Linux only once. The measured availability of former was 99.26% and 99.95% of the latter. This very interesting experiment showed, that GNU/Linux is much better in avoiding and containing hardware failures. The Reasoning[25] compared 6 implementations of TCP/IP – in GNU/Linux kernel, three as a parts of commercial general-purpose operating systems and two were embedded in commercial communication equipment. The company used automated tools to look five kinds of defects in code:

- memory leaks,
- null pointer dereferences,
- bad deallocations,

23 An article about mentioned test [13].
24 http://gnet.dhs.org/stories/bloor.php3.
25 The article about Reasoning test - [31].

- out of bounds array access
- uninitialized variables.

The results were at least surprising, because the embedded systems should have the best possible implementations of such protocol due to fact that their main aim is to enable communication. In GNU/Linux kernels 81,852 lines of source code (SLOC[26]) 8 defects were found - (SLOC), resulting in a defect density rate of 0.1 defects per KSLOC. Three proprietary operating systems (two were versions of Unix) had between 0.6 and 0.7 defects/KSLOC and the two embedded OS'es were 0.1 and 0.3 defects/KSLOC. Of course some of these defects were not sensus stricto problems. In case of Linux 4 bugs have no effect on the running code. This test showed also how many bugs were repaired by given product developers – in case of GNU/Linux it was only one bug while proprietary implementations have 235 ones. It gave repair defect rate of 0.013 and 0.41 defects/KSLOC. The CEO of Reasoning - Scott Trappe explained this result by that the FLOSS model encourages users not to report bugs – as it happens in case of proprietary software – but tracking down and fixing them, developers works in different way too – more on that matter you can find in chapter 4.2. Another factor, which influences reliability, is code quality. The 'Communications of the ACM' has published article[27], in which authors recap their study of 6 millions lines of source code of several programs over time with use of maintainability index[28]. Recapitulating results of their study they made a statement that FOSS

"code quality appears to be at least equal and sometimes better than the quality of [proprietary software] code implementing the same functionality. [...] May be due to the motivation of skilled OSS programmers[...]. Code quality seems to suffer from the very same problems that have been observed in [proprietary software] projects. Maintainability deterioration over time is a typical phenomenon... it is reasonable to expect similar behaviour from the OSS projects as they age."

26 SLOC – Source Lines Of Code; KSLOC – thousands of Source Lines Of Code.
27 Samoladas, Stamelos, Angelis, Oikonomou – "Open Source Software Development Should Strive for Even Greate Code Maintainability" ([26]).
28 maintainability index has been chosen, by Software Engineering Institute, the most suitable tool for measuring the maintainability of systems.

The results of Netcraft's ranking[29], done during November 2005, of the most reliable hosting companies shows, that FOSS solutions answer the commercial purpose. In the first ten: three GNU/Linux and three FreeBSD web sites, two on Windows and one on Solaris were found (one of sites which qualified to top ten was set on unknown server). It may be interesting to take into consideration a report of longest running systems by average uptime[30]. Here is the recap of the longest running systems focused on operating systems and web servers:

Operating system	Web server	No. of occurrences in the top 50
BSD/OS	Microsoft-IIS	3
BSD/OS	Apache	24
FreeBSD	Apache	16
Windows	Microsoft-IIS	3
Windows	Apache	1
IRIX	Apache	1

Table 1: OSes and webservers in the top 50 of the 'Sites with longest running systems by average uptime in the last 7 days (generated on 22nd march 2006)
Source: www.netcraft.com

The real-world reliability can be described using three factors:

- solutions has to be delivered predictably and consistently;
- reliable services has to be delivered efficiently;
- one can keep control over complex environment and 'keep it simple.

In November 2005, Security Innovation published 'Reliability: Analyzing Solution Uptime as Business Needs Change' – the report founded under the research contract from Microsoft. They have compared Microsoft Windows 2000 and SUSE Linux Enterprise Server 8. During 12 months they simulated

"*the evolution of e-commerce company, that has been changing the business requirements while continuing to maintain security through patch application. At the end of the [1 year] period, both systems are then transitioned to the more recent*

29 The Netcrafts reports - [18, 19].
30 Discussed results descend from report generated on 22nd March 2006 http://uptime.netcraft.com/up/today/top.avg.html.

versions of their respective operating systems, Windows Server 2003 and SuSE Linux Enterprise Server 9. Security patches were applied in 1 month increments, while new business requirements appeared at three month intervals. The experiment was conducted by three expert Windows administrators on the Windows side and three expert SuSE Linux administrators on the Linux side."

Results show clearly, that Linux solution does not meet business requirements because of too complex components dependencies which made 'painless' update impossible. Grater number of patches that were needed to be applied to GNU/Linux (187 to 39 for Windows) cause delays and in some cases administrators were unable to complete upgrade due to cascading packages dependencies. Third thing that made GNU/Linux unreliable for business solutions is, that the administrator – the only one who was successful in meeting all requirements – used components which were not directly supported by system vendor (SUSE/ Novel). Report says: "while the configuration did meet the functionality requirements, the administrator is now *on his own* to resolve potential future system failures." Let's forgot that this experiment was sponsored by Microsoft, and assume that it was perfectly unbiased. There are three arguments, that intrude while reading summary of this report:

- Assuming that GNU/Linux administrators were experienced ones it is quite interesting, why the Internet news groups and support boards were not full of questions about applying patches. When I was reading Security Innovation document, it was my first thought – why all users and administrators do not complain over and over again of upgrading their systems.
- Another argument is connected with time devoted on patching. GNU/Linux servers were always praised because of possibility of upgrading even the most substantial components of system without the necessity of rebooting machine and the number of patches was always treat as advantage of system which is continuously improved and fixed.
- Last argument is an answer for objection, that administrator becomes 'on his own' because he use third party components during update. Administrators have always community support and using third party components when they are more useful than 'official' ones is one of the main ideas of Free Software.

This experiment and its' authors conclusions shows only that to a certain degree Free/ Open Source Software and proprietary software must not be compared because applications of these two types have absolutely different origins and design philosophy.

4.1.2 Performance

The performance is not the most important feature of business application. Customers at first want business processes to be carried over in correct ways . Only when they are sure, that everything is as it should be, then they focus on performance issues.

> ***Performance:. The way in which a machine or other thing performs or functions: behavior, functioning, operation, reaction, working.***
>
> ***The Software Engineering Institute- Terms Glossary***

Performance benchmarks are very sensitive to the assumptions and environment, so the best benchmark is one set up to model given setting. I could again mention that it is much easier to find the bottlenecks in running system with full access to the source code, but instead of that I will focus on other FOSS advantages that make application much more effective keeping the high reliability. The strong argument that this is the right approach is that real performance boost can be obtained only when all parts of given system are fully compatible with each other and developers lay stress upon proper parts of application. The right proportions in the performance to maintainability trade-off can be easily lost -during application tuning, for example clear code which can be easily modified is much more important in any GPLed application, than hyper effective algorithm which is comprehensible only for small number of coders. Even in proprietary projects code maintainability is often more important than it's productivity – besides in most cases the bottlenecks are found not in the core algorithms but there, where data is transferred between different media (like selecting data from database before it is used in program flow). Here are some examples of how FOSS solutions prove that they can fit excessive business requirements pertaining performance:

- in February 2003 scientists broke the Internet2 Land Speed Record using GNU/Linux[31] - they sent 6.7 GB of data from California (using Red Hat Linux) to Amsterdam (European team used Debian GNU/Linux) at speed of 923

31 Dean Katie - "Data Flood Feeds Need for Speed" ([3]).

megabits per second in 58 seconds. One can say that it was not because they have been using GNU/Linux distributions, but the fact is that record has been broke with use of Free Software;

- benchmarks comparing Sun Solaris x86 and GNU/Linux found many similarities except web operations where GNU/Linux has obtained two times better result than Solaris[32];

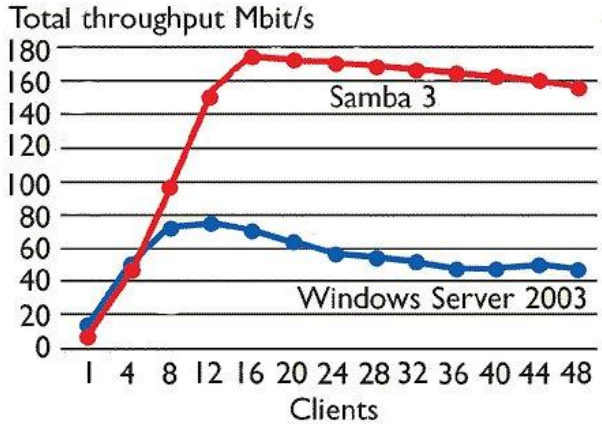

Illustration 8: Samba outperforms Win2003.
Source: IT Week

- In IT Week article - Samba 3 extends lead over Win 2003[33] - from October 2003 Roger Howorth recapitulate results of tests run by IT Week Labs which shows that the then version of Samba[34] "has widened the performance gap separating it from the commercial Windows alternative". In overall Samba 3 were performing 2.5 times faster than Windows Server 2003 and could handle at least 4 times as many clients as Microsoft solution before performance began to drop off.

32 Bourke Tony – "Sun Versus Linux: The x86 Smack-down" ([2]).
33 Howorth Roger - "Samba 3 extends lead over Win 2003 Roger Howorth" ([12]).
34 Samba is a free suite of programs which implement the Server Message Block (SMB) protocol – in other words it is a file and printing server software. The home web site of Samba project is: http://www.samba.org.

4.1.3 Scalability

According to the definition scalability tells us how well a solution to some problem will work when the size of the problem increases.

> ***Scalability:. the ease with which a system or component can be modified to fit the problem area.***
>
> ***The Software Engineering Institute- Terms Glossary***

The simplest example of estimating given application scalability is giving an answer to the question if given software package/ system will work if the project/ business become larger. Investing in cheaper system destined for operating in business environment (where requirements are not relatively high) is very dangerous, because along with the system's growths demands are growing too – in most cases meeting new requirements without costly modifications is impossible. Switching to completely new system is not a good idea either – process of migrating data and training all employees is expensive and risky. Some scalability issues are connected with optimising source code for given platform or adaptation and in this area Free Software is unbeatable. Obvious example is growing company which main systems can be modified without changing for example GUI or functionalities which are not affected by changes in business activity. Another mentioned instance of scalability problem is porting applications with the same core code to completely different hardware, when functionality is one thing but the cost of software in case of many types of devices is a very peculiar issue. The most universal software package I have ever heard about is GNU/Linux. There are one-floppy versions, which are used as routers software. GNU/Linux can be run even on old, obsolete hardware – there is no need to use the cutting edge hardware technologies for many purposes. Of course such machines – old or new – do not have to be PCs. GNU/Linux and NetBSD have been already ported to over a dozen different chipsets (like x86, Intel Itanium, ARM, Alpha, IBM AS/400, SPARC, MIPS, 68k, Power PC). GNU/Linux is commonly used for parallel processing with use of Beowulf architecture[35]. Even the very process of developing FOSS can scale to develop large systems. The RedHat Linux 7.1 has over 30 million SLOC – it stands

35 Beowulf – (1) the legendary hero of an anonymous Old English epic poem composed in the early 8th century; he slays a monster and becomes king but dies fighting a dragon;
(2) multi-computer architecture which can be used for parallel computations. Frequently composed of one tie-server.

for 8'000 man-years and over 1 billion USD to implement this distribution.

4.1.4 Security

Quantitatively measuring security is very difficult. It is said that FOSS systems are often superior to proprietary ones

> ***Security: the ability of a system to manage, protect, and distribute sensitive information.***
>
> ***The Software Engineering Institute- Terms Glossary***

Before I compare the security aspects of FOSS and proprietary solutions I would like to mention an example of Coca-Cola company, which has the very secret recipe of preparing their soda. The interesting thing is that that formula is not secured by any patent. The security if the Coca-Cola company is successfully protecting it with use of internal procedures since 50s. The Coca-cola recipe is an example of the fact that commonly used procedures are not necessary the best possible. What Coca-Cola has done was resolving that if recipe must remain as secret nobody should know it. Software developers deal with similar situation – the commonly used technique is to hide security bugs from public as long as it is possible and fix (or not) in the meantime. The problem with hiding such bugs is that even they were not found by testers, they can be found during everyday usage or crackers[36] attack and exploited, it is just a matter of time, after that all application system users are exposed to attacks and losses before – if ever – appropriate patch becomes available. It would not be a problem, if only software developers found vulnerabilities before people who are interested in breaking security on a given system. Unfortunately in most cases such vulnerabilities are found after, or rather because of successful attack. In my opinion testing is biased because of the fact, that tests are prepared by the same people who design and develop programs. Nobody can require they think out situation which in fact code they have designed or wrote does not support. It cause that proprietary software has never an undetected error. Everyday practice gives enough feedback to conclude that trustworthy software developed in traditional – proprietary – manner is expensive and rarely customisable. FOSS is free from mentioned disadvantages just because of full availability of the source code. Bugs

36 Cracker - an individual who attempts to gain unauthorised access to a computer system. The term was coined ca. 1985 by hackers in defence against journalistic misuse of "hacker". (from definition in 'Jargon file' by Eric S. Raymond).

are found faster because more people can use programs/ systems and examine its code in the same time. Bugs are patched more frequently even without need of engaging main developers team. On 18th January of 2005 vnunet.com informed[37] about results of test made by non-profit IT organisation Honeynet Project which shows that it takes, on average, 3 months to break security of freshly installed GNU/Linux. Iain Thomson – the article author – remind that in equivalent tests which were run in 2001 and 2002 it took crackers only 72 hours. Result shows that distributions of GNU/Linux are ready to use after installation without fear of systems being compromised, in contrary 'fresh' – unpatched – version of Microsoft Windows lasts only few hours.

Symantec Internet Security Threat Report published on March 2006[38] compares some proprietary and non-proprietary packages in context of security. Below, in tables, I've included some results of Symantec's tests.

Configuration	Average time needed to compromise
Microsoft Windows 2000 Server - No patches	1:16:55
Microsoft Windows 2000 Server – Service Pack 4	1:32:08
Microsoft Windows 2003 Web Edition – No Patches	4:36:55
RedHat Enterprise Linux 3 Web – Unpatched	Not compromised

Table 2: Time to compromise web servers.
Source: Symantec Corporation

Configuration	Average time of compromising
Microsoft Windows XP Professional	1:00:12
Microsoft Windows 2000 Professional – No Patches	1:03:18
Microsoft Windows 2000 Professional – Service Pack 4	4:36:55
SuSE Linux 9 Desktop	Not compromised

Table 3: Time to compromise desktop computers with firewalls deactivated
Source: Symantec Corporation

37 http://www.vnunet.com/actions/trackback/2126530.
38 Symantec Internet Security Threat Report - Trends for July 05 – December 05 – Volume IX ([29]).

4.2 *Software development*

The development is the process of analysis, design, coding and testing software, some will say that development includes creating documentation too. The more popular project is, the more people contribute their work to it and the very process of development becomes much more different from 'traditional' – proprietary software development standards. Common factor in definitions of software engineering is, that systems are built by teams and such team participants must be able to communicate. Another factor is the need of documentation necessary to develop, use and maintain programs. A single developer or small team which decide to create new piece of software covered by one of GPL compliant license can decide to use any of life cycle model[39] that can be effectively applied. When the project begins to attract new participants[40] it is impossible not to introduce communications channels and developing paths, which secures progress in project development. When ideas and new code start to come from outside of core development team, most of the simple collaboration methods become insufficient. In some cases it is easy – there are methodologies developed to be used during FOSS life-cycle, some non-profit organisations – like sourceforge.net - facilitate the process of collaboration. However not every application should be developed in 'full bazaar mode' when everybody can contribute and main stream of development is based on the community members collaboration. Especially business and science projects shouldn't be treated this way – the requirements which follows the agreement between developers team and the orderer, have the highest priority, but may undergo a change under influence of 'outside' ideas. For example, if the project development process was controlled by official project documents, new types of those should be created. No matter which type of life-cycle given project represents, some changes have to be led in respective phases. First and the most important modification, which must be done, is switching to any incremental or quasi-incremental[41] development model to simplify implementation of new ideas, features and requirements modifications.

39 More on life-cycle models can be found in "Software Requirements: Analysis and Specification" by A.M. Davis ([4]) and "Requirements Engineering: Processes and Techniques" by G. Kotonya and I. Sommerville ([14].

40 By participants I mean developers and users who contribute to the project in any way possible (coding, giving a feedback on bugs, coming with new ideas, etc.).

41 For example if project was developed during waterfall process at least feedback loops should be added.

4.2.1 Advantages and features of dualism and parallelism in FOSS development

As I have already mentioned in some situations development process should be dual and parallel for the purpose of filling customer/ orderer requirements and at the same time to take advantage of community support and give the project chance to 'live its own life'. There are two main ways of creating software in such dual and parallel system:

- First one is used by some Open Source Software producers like Trolltech does – the official 'product' can be changed, patched etc. only by its vendor. Those who want to make a modification can create only plug-ins but not modify and redistribute the source code. Of course Trolltech developers do implement community members ideas and such plug-ins in official versions of their products and thanks to that it remain conformable to the company policy. Such procedures can't be introduced to GPL covered application because GPL secures freedom to redistribute source code.
- Second way of dual development is to develop one version of application which can be supplied to the orderer and second version which is developed without any restrictions. It seems not to differ from Trolltech's model described above but in this case community, core developers and their employer benefit from the very best practices of Free Software development without loosing touch with projects main objectives.

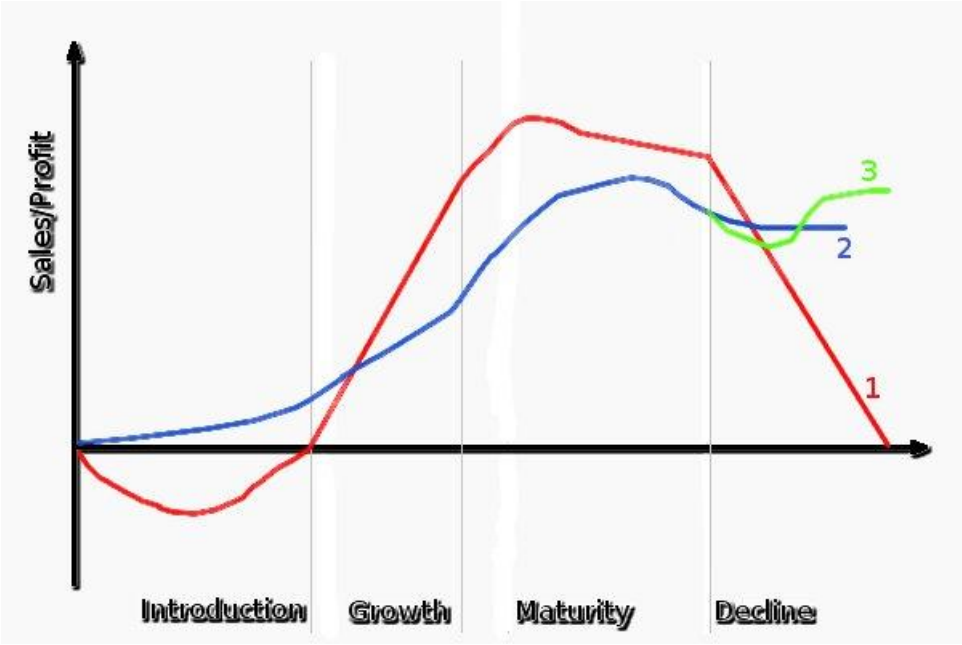

Illustration 9: Life cycle product

Curve no.1 – proprietary software project
Curve no.2 – 100% finished FOSS project
Curve no.3 – FOSS project, requirements redeveloped during decline phase

Source: Own elaboration

Now I will describe what activities should be incorporated into individual life-cycle phases during FOSS development. It's important to mention, that there are not required in case of standard development process but dual development won't be effective without them. The requirements analysis should involve the revision of ideas and features submitted by users and developers from outside of core developing group. It is highly possible, that someone will find better way of doing something what has been already coded, or will bring brilliant idea neither orderer not developers have thought about. Normally, after some iterations, project falls into decline phase where it can pass away or new requirements are set (and practically new development process starts). Thanks to FOSS developers approach it becomes more dynamic and it stays in the maturity phase because of its main features, which are added continuously. The blue curve shows add-ons state over time. Such approach is useful only when this phase is repeated in each iteration. If risk analysis is done during projects life those who make it, has to take into consideration, that additional functionality – acquired thanks to outside contributions – can increase developed application functionality. On the other hand, such additions could decrease application performance, it's level of customisation,

reliability - such synthesis should be a part of requirements validation phase. Assuming that this way added functionality is absolutely desirable during risk analysis, somebody has to decide if extending test area and implementing idea or even ready code into given project is cost effective. On the other hand, before unit tests developers should inculcate modifications which follows from testing done by users who gave feedback on that. Such testing should not replace internal unit and system tests because of different system/ application structure (but they are valuable source of hints on programs behaviour).

4.2.2 Factors facilitating collaboration in development process

The very process of development does not differ in case of dual-parallel and standard developing. There are some factors, like modularity, documentation, infrastructure, plan of releases, which make collaboration much easier. In 1962 Kristen Nygaard and Ole-Johan Dahl designed SIMULA 1 and created foundation of Object Oriented Programming (OOP). From that moment, the idea of creating smaller parts and building complicated system with use of them develops. This idea becomes popular not only in the field of programming, many system and applications are designed as set of blocks which are also called modules, plug-ins/ add-ons etc.. GNU/Linux kernel is created and developed as small core program and many additional modules, which face many functionalities –from hardware drivers to security issues. It won't be easy to develop such system as one program, when many people work on different parts of application/ system it is easier to create universal interface responsible for unification of system and its plug-ins than rebuilding whole system. System based on modules is easier to customize for end user and – what is the most important issue I think – it is easier to develop modularised application by programmers. If one wants to create new functionality has only to learn how modules work in given system and how they should be created, nothing else. It is much more difficult to create developers community around project on the stage, when it's development requires newcomers to learn how the system works as a whole and in details. The main disadvantage of managing modularised project is a must of verification if submitted modules really service new and useful functionality. The most widely known projects based on modules are GNU/Linux, OS Commerce, TYPO3, Apache, PHP, Magnolia, eZ publish, Cocoon[42].

42 Here you can find more information on mentioned projects:
OS Commerce – http://www.oscommerce.org ;

Documentation can improve collaboration. It is obvious – it answers questions that new developers and users ask. There are many types of documents which are commonly used in connection of FOSS projects: manuals, tutorials, feature lists, change logs, guidelines, end-user documentation, etc. Well documented application is easier to maintain and to develop. Users should be aware of fact, that there is project documentation, so they won't ask questions, which are already answered in prepared documents and they familiarize themselves with the project as a whole. Maintaining modularised code, creating and providing high quality documentation as well as maintaining the project need infrastructure, which enables communication between community members, makes application and documentation accessible for users, provides tools, the process of development and maintaining would be impossible without. Sourceforge, for instance, provides many services which are essential for FOSS projects, but the phenomenon of mature and successful projects moving to other collaboration platforms, often to ones created for given project, is perceptible. It follows fact that it does not matter if we are analysing proprietary or FOSS project development – the more complicated project is, the more 'sophisticated' tools are need by it's developers. Last factor I want to shortly described is a release management. I have noticed two main approaches dominant on the FOSS market – of frequent releases and one representing theory of reaching huge milestones at the end of each cycle. A good example of the latter is one of GNU/Linux distributions – Debian. New releases were often postponed because developers want the end product to be stable and reliable as much as it can become nearly 'bullet-proof' – Debian is the only GNU/Linux distribution which has the ISO certificate and it is said that is the best one on the market. I can not say that such policy is not good, but too long periods of time between releases, may result in market share loss. Of course during three years between two last Debian releases continuous support for previous version has been maintained, updates for all components were kept, new applications were included to test and development releases. But on the other hand, publishing new, stable and 'complete' release has greater influence on both current and potential users as much as on developers and marketing &

TYPO3 – http://typo3.com ;
Apache – http://www.apache.org ;
PHP – http://www.php.net ;
Magnolia – http://www.magnolia.info ;
eZ publish – http://ez.no;
Cocoon –http://coccon.apache.org.

social impact on the market. The contrary approach can be found in many minor projects, which are released with new version number after each patch or even cosmetic modification. Such policy helps to control changes and keeps users up to date with current version, but can easily discourage and/or bore them. The PHP[43] is an good example – security patches were published often (at least they were in early releases, now there is no need to do that) but releases of completely new versions (like from 4 to 5) are done only due to introducing considerable functional changes and improvements.

4.2.3 What kind of people and why do develop and maintain FOSS projects

The community of people who contribute their work to Free and Open Source Software projects is not composed of good Samaritans, who spend their time just to create tools, which can be used by individual and institutional users. Developers and users, who often have share in development process, have their own motivations and aims. Some are intrinsic but most are not. To describe contributors motives better, I am going to describe the structure of the community. I base my conclusions on results of 'Survey of Developers'[44] published on June 2002 by The International Institute of Infonomics[45].Most of developers who are share of the sample is younger than 27. One third was between 16 and 21 old and one third was between 21 and 25 age old, when started developing FOSS.

43 More info on PHP can be found at http://www.php.net.
44 Survey was done between February and April 2002 in cooperation with Berlecon Research GmbH (Berlin). More info and the original report can be found on http://www.infonomics.nl/FLOSS/report/.
45 The International Institute of Infonomics is a department of University of Maastricht (The Netherlands).

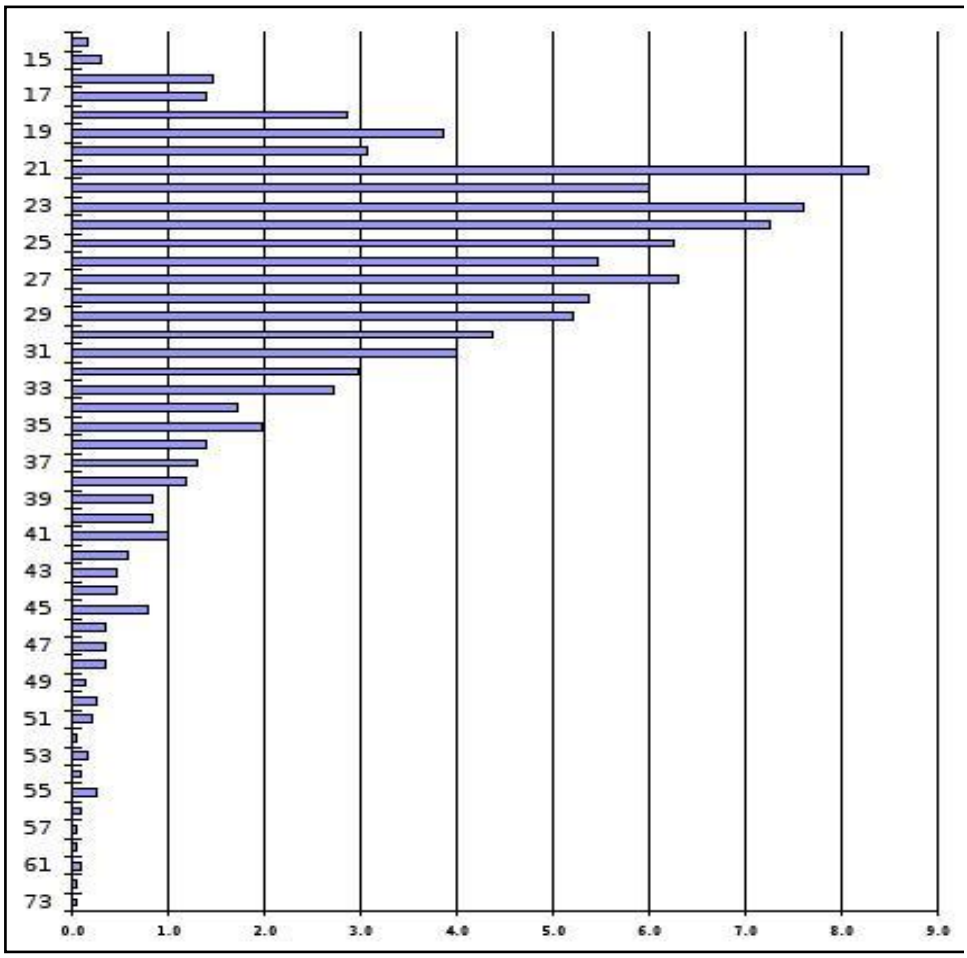

Illustration 10:Curent Age of OS/FS Developers
Source: Free/ Libre and Open Source Software: Survey and Study. Part 4: Developers

Another thing is that 60% of developers live in a kind of partnership but only 17% of developers have children.

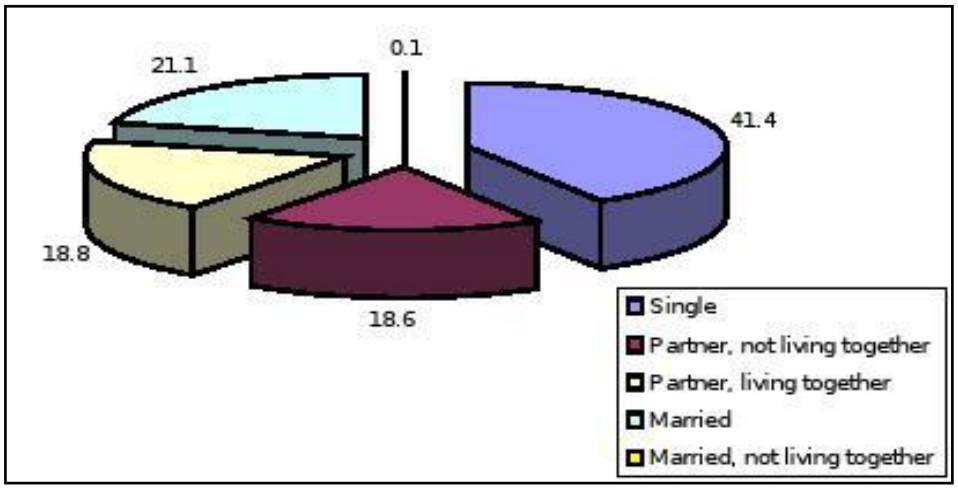

Illustration 11: Civil status of OS/FS Developers
Source: Free/ Libre and Open Source Software: Survey and Study. Part 4: Developers

To try sketch the possible motives which follow current behavioural standards, I need to describe educational

Level of education	share of the sample (%)
Elementary school	2.0
High school	17.0
A-Level	8.0
Apprenticeship	3.0
University – Bachelors	33.0
University – Masters	28.0
University – PhD	9.0

Table 4: Level of education of FOSS Developers
Source: Free/Libre and Open Source Software: Survey and Study. Part 4: Developers

and professional (employment) background of FOSS developers

Profession	share of the sample (%)
Software engineer	33.3
Engineering (other than IT)	3.2
Programmer	11.2
Consultant (IT)	9.8
Consultant (other sectors)	0.6
Executive (IT)	3.2
Executive (other sectors)	0.3
Marketing (IT)	0.2
Marketing (other sectors)	0.0
Product sales (IT)	0.1
Product sales (other sales)	0.0
University (IT)	5.0
University (other sectors)	4.3
Student(IT)	15.8
Student(other sectors)	5.1
Other (IT)	5.2
Other (other sectors)	2.7

Table 5: Professional structure of FOSS developers
Source: Free/Libre and Open Source Software: Survey and Study. Part 4: Developers

The group of employed is about 79% (65% are employed and 14% self employed) of all developers questioned during the survey. Students are the separate group of 17% of all developers and are not treated as unemployed who are in the 2% share. Another group of 2% are those, who are changing the job – generally not working at the moment when they were asked. This results show, that people who spend their time working on FOSS projects do not perform that because they have to, or they do not know what to do with their spare time. The level of education and employment field of most of FOSS developers indicate, that they are finding some values they have not discovered in private/ professional life. The answer for question "why somebody, who is a software engineer, devote some of his free time to non proprietary projects – in most cases not being paid for that?" can be found in recap of surveyed developers expectations from other members and community as a whole and reasons they join and stay in FOSS community for.

Reason	share which join (%)	share which stay (%)
learn and develop new skills	**78.90**	**70.50**
share knowledge and skills	**49.80**	**67.20**
participate in a new form of cooperation	34.50	37.20
improve OS/FS products of other developers	33.70	39.80
participate in the OS/FS scene	30.60	35.50
think that software should not be a proprietary good	30.10	37.90
solve a problem that could not be solved by proprietary software	**29.70**	**29.60**
improve my job opportunities	**23.90**	**29.80**
get help in realizing a good idea for a software product	**23.80**	**27.00**
limit the power of large software companies	19.00	28.90
get a reputation in OS/FS community	9.10	12.00
distribute not marketable software products	8.90	10.00
make money	4.40	12.30

Table 6: Share of developers who join and stay in the FOSS community because of particular reasons
Source: Free/Libre and Open Source Software: Survey and Study. Part 4: Developers

Most of developers do they job for different reasons than money and these reasons can

be divided into two types – intrinsic and extrinsic. The most important determinants should be considered from the point of view of developers expectations and their origins. The most of developers claim, that they do participate in FOSS community because they want to learn and develop new skills. Two most numerous groups are the software engineers and IT students – they can do FOSS for fun along with rising their skills level and at the end of the day they are improving current and future job opportunities or use newly acquired skills in they paid work. Solving problems that could not be solved by proprietary software is a very good example that the idea of bazaar developing proves true. Some task cannot be finished with use of software, which comes without source code, especially in the various science fields – it can be assumed basing on the educational and professional structure of the sample. 27% of questioned developers have joined the community to get help, on the other hand nearly 40% of sample claimed they contribute to improve others' programs – it cannot be said why members of former group need help either members of the latter one contribute their time to the FOSS projects[46], but both groups surely reach their aims (otherwise they would not stay in the community). Summarizing – the three most important determinants, which make people to contribute their work to Free/ Open Source projects are:

- intellectual curiosity[47],
- a need to solve developers own programming/ software related needs[48]
- promise of higher future earnings[49].

These determinants can be easily derived from the fact, that most of developers are young, educated man who are affiliated in some way to IT sector. Nearly 60% of the sample is single or live alone – this figure gives partial answer to the question why 35% declare, that they stay in FOSS community to participate in it. Cristina Rossi and Andrea Bonacorsi in their elaboration[50] point at 'fun factor' and ego boosting incentives accompanying the process of developing software. Two of them is the sense of

46 Lerner, J., Tirole, J. - "The Economics Of Technology Sharing: Open Source And Beyond" ([16]).
47 Boston Consulting Group - "Boston Consulting Group/OSDN Hacker Survey" ([1]).
48 Lakhani, von Hippel - "How Open Source Software Works: 'Free' User-to-User Assistance" ([15]), pp 923-943.
49 Free Software Foundation - "GNU Public License" ([10]).
50 Rossi, C. and Bonacorsi, A. - "Intrinsic motivations and profit-oriented firms in Open Source software. Do firms practise what they preach?" ([25]).

belonging to the community which is based on (mentioned in chapter 5.3) gift culture and economy and collective identification enhanced by presence of an enemy – the proprietary software producers and vendors. Figures in Table 6 demonstrate, that such factors play a substantial role in developers view of private and professional world.

4.3 *Comparison table*

	proprietary software	**non-proprietary software**
error detection process	Done during preprepared test phase, based on test scripts and test data. Some test are done by the beta testers who are not directly connected with the developing company but this method bear different security matters. Taking into consideration number of bugs in end versions this method is untrustworthy.	Bugs are detected during testing phase but tests are mostly done by persons who are only interested in getting 100% working application. Test are done with use of data with is highly probable to be the real – 'production' data. Proven reliability of FOSS products shows this method efficiency.
bug fixing after testing phase	Can negatively affect sale of given product and other products of the same producer. The cost of a patch can be high depending on magnitude of patch and the method of main product distribution.	Can be done even by the end user, patching or releasing new version is not connected with additional cost (except of software created in result of an agreement which covers case of handling bugs detected in '100% working' versions.
end user availability to foresee application behaviour in changing business environment	Based only on experience of other users, but some licences forbid publishing negative opinions about covered product.	Product's scalability can be verified with use of test paths – which can be well prepared due to source code availability. Users can easily obtain knowledge on the cost of widening application functionality and

	proprietary software	non-proprietary software
		implementing additional features.
methods of preserving end user security	The source code and features responsible for security are protected from unauthorised access, which simply means that no one can make use of them.	The whole source code is available to anyone who want to make use of it. Thanks to that bugs are detected faster because users can not only test application in everyday usage but also analyse the source code and test chosen pieces of code.
sources of ideas during and after main development processes	Implemented features can follow particular customer needs or needs of possibly the most numerous group of potential customers. Once the product is released implementing new features is connected with cost bore by either producer or customer.	Ideas can be implemented without regard to development phase by project main developers or developers/ users who need given feature.
methods of preserving end user interest in the product	End user must get what he/ she wants but some functionality/ features must be always missing to tie customer to the software vendor/ maintainer for as long as it is possible.	Improving software in addition to be always step before end user demands or bring project to the end when all functionality, which can be needed is implemented and 100% working.
methods of encouraging developers/ reasons developers do develop	Mainly financial profits.	Mainly to learn and develop new skills, share knowledge and skills, improve job opportunities.

Table 7: Comparison of proprietary and non proprietary software
Source: Own elaboration

5 FOSS based business model – rules and strategy

5.1 Philosophy

Along with increasing availability of software as a tool, it becomes less differentiating for business. Bruce Perens[51] described two forms of technology used in business: differentiating and non-differentiating[52]. Software is mostly used as a tool which makes companies more efficient and in most 'fields technology non-enabled' companies simply wouldn't have a chance to make any business. As long as company is not a software manufacturer its customers do not care what office suite or database engine company use. It does not matter for one who runs business if competitors know what software tools he use because such tools – from 'off-the-shelf' software packages to software created during in-house or contracted development process - are available for everyone. That is why paying each time to buy different software package for the same functionality is pointless. Software development process is different from any other. One who works in building, can not copy and paste foundations, even if during the construction identical buildings are going to be build. The very same problem occurs during proprietary software development. Imagine, that a designer need two different graphical tools, just because one has couple features not implemented in the second one and contrariwise. Excluding that couple of distinctive features first package parallel the second, and at the end of a day company, which employs that designer pays twice for most of the functionality. Another thing is the must of creating suitable environment to run mentioned two products – simply by paying for the operating system. The most popular one includes some tools needles for a designer, but one can not buy it without them. Modern business came to the point, when companies pay many times for the very same products, pay for unwanted software, pay for another software tools to cut some functionality in others, pay for employees' trainings and an the end: pay people to maintain systems which grows larger and larger just because of proprietary software vendors strategy and market philosophy.

51 Perens Bruce - "The emerging economic paradigm of Open Source" ([24]).

52 The software have differentiating role when it does make an economical or strictly business difference if the competition use the very same software package. Software tools chosen by given company influence its' competivity. An example of differentiating software tool can be a scoring system used by creditors. The non-differentiating role has software which is used for general purposes, like – for example – text editing. At the end of the day it does not matter which office package is used – end products – documents will be the same.

The Open for Business Project philosophy mentions rules which enables 5 'E's':

1. Ease of Cost,
2. Ease of Installation,
3. Ease of Customisation,
4. Ease of Integration,
5. Ease of Use.

Rules presented above summarize briefly the main ideas of FOSS. Sharing the code enables developers to collaborate during processes of software creation, usage and review. Not only code is shared, but ideas and overall knowledge on development tools too – it means that customers and users can give feedback to software creators to get useful product and programmers can minimize the code size and work needed to implement given functionality. Use the main idea of sharing the software and not reinvent the wheel but reuse existing solutions and components in new applications. The conclusion is: the best way to satisfy the biggest group of software users and developers is to concentrate on what everyone wants and do everything to get it as cheap and fast as it is possible.

5.2 *Risk management and TCO*

When choosing right software for commercial usage one should be concerned in production factors Total Cost of Ownership (TCO). The cost of maintaining software, training users, customizing 'off-the-shelf' packages is often much higher than the cost of buying it, or hiring contracted developers to create needed applications. There are two most common paths, which company can choose while selecting software – use of 'off-the-shelf' software and hire developers to create needed applications. Both bring different problems and expenses. The 'off-the-shelf' software represents about 25% of all software development[53] that is why it has to satisfy needs – be capable of serving a many different purposes well - of the largest possible group of customers to be profitable for its manufacturer – which/who usually bore whole cost of development of such package. In fact only 10% of typical package meets the need of customer, residual

53 In that report [21], "Packaged Software" represents 24.6% of the industry. All other industry sectors that represent computer programming, including all of Computer Programming Services, Computer Integrated Systems Design, Computer Processing and Data Preparation and Processing, Information Retrieval Services, Computer Facilities Management Services, and some sub-categories of Software Publishing represent the remainder.

part remains useless, but greater functionality means making work given application for a particular purpose more complicated. The latter path does not bring in such problem – ordered software includes only the very needed functionality and process of training users is much less complicated, because stuff have to know only how to use implemented functions and do not need to know which part of functionality should not be used. Such customized software restricts the number of distractions during every day work and possibility of problems caused by superfluous functionality. In-house/contracted projects are efficient in about 70% but the risk of project failure reaches 50% - invested money can never bring any profit. On the other hand, this is the only way of acquiring software, which protects customer differentiation. When company decide to choose solution based on FOSS, the most efficient way is to customize existing FOSS application. The superiority of such solution is obvious – it's much faster and cheaper to pay developers only for changing source code, than for creating every feature from the very beginning. Another important factor that should be concerned, before company decide to use given software package, is application maturity. There is no difference if it is proprietary or non-proprietary tool – more mature means lower risk of bugs, implementing/customizing failure, better support. Because of the fact, that FOSS application development is different from other types of software, development some indicators of maturity may differ. Such indicators can be found on www.sourceforge.net[54] how long piece of software is in development, how many project leaders and developers are working on in. Activity of project developers and feedback that they give to the users community is very important. Users activity – for example on projects boards – can be use to estimate projects popularity to check, if the application meets users requirements in real life situations. Project maturity may be considered as application quality – more numerous and active developers and users group means better support and faster development which means higher reliability, better performance and more secure end product. In terms of proprietary software such characteristics would indicate quality level of application. Next aspect of software, which makes it useful, is compatibility. Proprietary applications are often certified to be

54 SourceForge.net is the world's largest Open Source software development web site, hosting more than 100,000 projects and over 1,000,000 registered users with a centralized resource for managing projects, issues, communications, and code. SourceForge.net has the largest repository of Open Source code and applications available on the Internet, and hosts more Open Source development products than any other site or network worldwide. SourceForge.net provides a wide variety of services to projects we host, and to the Open Source community.

compatible with one another. Smaller producers often make their software in cooperation with huge ones just to make sure that their product is fully compatible with other applications or operating system. The open standards used in the world of FOSS development make non-proprietary applications more interoperable than its proprietary equivalents – in most cases such fact won't be formally certified. It means, that before choosing particular package, it should be precisely checked, if it meets requirements related to compatibility and interoperability. When company acquire needed software, the question is how much it will cost from the moment of application selection. Software have to be installed, users trained, problems which occurred during everyday usage solved, requirements may changed that leads to two essential for altering original code questions: documentation and source code. In terms of implementing FOSS application institution should make sure that source code is available in form, which enables developers to change it easily – properly formatted and commented. Whatever company is going to customize application by changing the source code or not chosen application should be well documented. In spite of common view that hackers[55] do not like to write documentation most of FOSS programs come with a lot of documentation. This is an effect of developers will to make their code and application useful and popular. Another positive aspect of well documented source code is lower risk in situation, when developers abandon their project and nobody will decide to continue their work. The main costs of maintaining FOSS application (cost of support, training, customisation, lack of contingency) can be easily reduced just by use of well known good programming practices (like keeping integrity of code and documentation, and participating in developers and users community).

5.3 *Use, market and monopoly value*

Software, like any other product, has its value. In terms of economy, we can distinguish three types of value: use, market and monopoly. By definition the monopoly value is the one, which is set by the sell party, which has exclusive control of given good market. The monopoly value of software is the indicator of application value generated when given application is unavailable for competitors. As I have mentioned, software as a tool has non-differentiating role in business - as long as software manufacturing is not

55 Hacker – [Jargon File v0.4] - One who programs enthusiastically (even obsessively) or who enjoys programming rather than just theorizing about programming.

considered – almost everyone has access to commercially used applications. The monopoly value of such tools is convergent to zero, only in-house or contracted project give advantages of monopoly value, but cost of such solutions and risk connected to them is very high. Another type of value is the market one, which tells customers what is the real value of the product on the competitive market – the market with more than one seller and buyer in it. Competition – at least in theory – brings to situation when every products' price is placed in the point, where curves of demand and supply intersect. Of course every seller want to earn as much as it is possible and each user rather get needed software without a charge. That leads us to the use value of product – the economic value of application as a tool that means "the amount of money that is considered to be fair equivalent for something else". The problem arise when customer pays for the software package which is in 90% useless for him/her – popular applications are not sold for 10% of their real value - adapting such strategy would lead to bankruptcy. Customer is the one who covers all lids, which follow inconveniences of retail software production.

The main idea of every cost management theory is to cut costs everywhere where it is possible. So why to pay for the generally accessible software which in approximately 90% consist of unwanted and - in fact - useless functionality, or why anyone should invest money and time into project which can fail with 90% probability. If most of the software have got zero monopoly value why not to cooperate and share cost of creating application – like many companies do. From the customer point of view a lot of software only supplement different product and buyer do not want to pay for it, so supplier bore the whole costs of creating such program – the best example are hardware drivers. The consequence is higher price or less quality of main product and in fact the one who pays is customer. First solution is to create some software in cooperation with another company – like IBM and DELL do – but it brings some difficulties which cannot be resolved without fund flows, so at the end of the day product costs less but it could cost much less if some new factors – like increased spendings on logistics - wouldn't arise. Fact, that software became a generally available tool, should be a consolation for its users. Because of factors mentioned in this chapter and intricate legal issues it is not. FOSS reveal as a remedy for the most matters, which gnaw milieu of institutional software users. There is no monopoly, nor market value of Free/Open

Source Software, but the costs generated by these values does not exist neither. One can call zero monopoly/market value a price, which is pay by FOSS users – in terms of free market this 'price' is pay by producers of proprietary software. Considering given application the most important question is how much users save and producers lost due to using Free and Open Source Software by former ones. If theoretical market value of FOSS would be taken in consideration users save much more than producers/developers and vendors lose, but we can't do such calculation – rules of developing and vending proprietary software make it impossible to create software of such great merit as FOSS distinguish itself. Full customisability make Free Software the best solution for companies which do not want to spend money on features they do not need. Every GPLed tool can reach the highest utility level in given conditions thanks to possibility of changing and redistributing application source code. The effective decision can be hiring developer who will be responsible for adapting existing FOSS application to institution's needs. Such programmer have only to change existing source code, add some new functionality, but time and money are not wasted for coding things that were coded by somebody else. Using application, which is coded and tested by somebody else, is the fastest way to create own program. Good idea is to redistribute changed application, or contribute developer's changes to given program users/developers community. Besides of putting something back into proverbial pot there is huge chance that somebody will use, test, improve 'our' code and make it available the same way we did. Earlier our solution becomes generally available grater is the chance for attracting more users and developers - this is the first mover advantage. The first company which GPLs their software has incomparable advantage over it's competitors. More developers who are interested in developing certain project will work on it and more stick to it for good. Why it is good when the biggest possible group of people is involved in a project I will discuss in chapter 6. Giving a summary of FOSS economical value is simple. It's features cause lack of monopoly and market value – because of being generally available in most cases for zero cost - but also create a phenomenon of product which can became very effectual. FOSS effectiveness comes from users freedom of sharing ideas, code and contributing work to the very same project. A sample company can save – and even profit - in couple realms by switching to Free Software. Firstly FS and most of OSS can be downloaded from the Internet free of charge. Full customisation – even achieved in result of employing programmer – makes much more effective and cheaper

training of users and application usage. Instead of buying couple of programs institution can pay only for implementing some features into existing code. With good approach that effect can be gained without spending even a single penny – making developers and programmers interested in project can create a community that will develop and support software used in sample company.

5.4 *Differences in products life cycle*

In 80's and 90's situation on software market caused growth of FOSS popularity and hasten its evolution. Now Free Software is so common and popular, that some roles in business model have changed. Different product structure means different structure of its producer and vendor. Let's have a quick look at typical product's life cycle phases:

	proprietary software	**non-proprietary software**
INTRODUCTION PHASE		
Price	High due to customers will to pay premium for possibility of using new product.	Zero or fixed in case of contracted or in house development processes.
Promotion	Limited and focused on attracting specific group of customers.	Focused on attracting only those who need the given product or those who will find interesting contributing to the project. In case of contracted/in house project limited.
Distribution	Direct or limited.	Through channels known by potential users/customers/developers
Sales	Limited to small team of highly skilled salesmen with good knowledge of market.	Unlimited - product is available on the Internet. In case of contracted/in house project it might be limited.
Development	Focus on time to market and uniqueness.	Focused on adding new features, extending functionality of existing ones and bug fixing

	proprietary software	**non-proprietary software**
Manufacturing	High expenditure for new production capacity.	Non applicable
Support	Direct factory support. Engineering involvement is required.	Mainly technical, the aim is to attract new developers.
Training	Focused on new product features, benefits, differentiation, pricing and functionality.	Focused on showing new users benefits from using the product and receiving the feedback – opinions on new product
Technology	New and innovative.	The one – covered by GPL or GPL compliant license - which suits given enterprise the best.
Competition	New and innovative. Limited. May be offering different solution for the same problem or application.	There may be no competition, or existing solution or its part can be used in new project or became a external part of it.
Market share	Low overall.	Low overall
GROWTH PHASE		
Price	10% of market level. – 10% if the brand name is weak and competition is severe, + 10% if sales are good and competition does not have similar product to offer.	Zero or fixed in case of contracted or in house development processes
Promotion	Heavy. Targeted promotions, trade shows, direct mail, sales seminars, articles and press releases.	Articles covering product's functionality, focused on informing potential users that such application has been recently made available.
Distribution	Highly skilled. Focused channels with strong technical skills	Through channels known by potential

	proprietary software	non-proprietary software
	if needed, complementary products and services.	users/customers/developers (the Internet, magazines)
Sales	Everywhere possible. Retail shops, telephone, Internet.	Unlimited - product is available on the Internet – may be found on more websites. In case of contracted/in house project it might be limited.
Development	Complete development. Market penetration is sustained with variations and improvements of the product.	Main and the most wanted features should become stable. Bug fixing and adding new functionality.
Manufacturing	Addition of capacity and automation.	Non applicable
Support	Phone support.	On the Internet – covers usage in general. First line in bug fixing process.
Training	Transition to newer version of product.	Includes forms of implementation into existing applications, usage in general and ways of developing.
Technology	Newer and leading edge.	The one – covered by GPL or GPL compliant license - which suits given enterprise the best.
Competition	New appearing worldwide.	No competition or competitors have different goals.
Market share	High growth. All out market warfare with competitors.	Increased mainly on private users market and FOSS experienced corporate users. Corporate users who want to switch from proprietary application covering similar functionality are waiting for

	proprietary software	non-proprietary software
		stable version which reliability has been proved true by the former ones.
MATURITY PHASE		
Price	Stable.	Zero or fixed in case of contracted or in house development processes
Promotion	Focused on reliability, quality, predictability, new enhancements.	Focused on reliability, quality, predictability, availability and usage statistics (covering individual and corporate users)
Distribution	Many distributors, alternative channels, offshore sales.	Through channels known by potential users/customers/developers (the Internet, magazines)
Sales	Direct sales focused on hi-volume, high profit.	Unlimited - product is available on the Internet – may be found on more websites. stable version are published in magazines. In case of contracted/in house project it might be limited.
Development	Focused on cost reductions.	Bug fixing and adding new functionality.
Manufacturing	Focused on increasing yield and productivity.	Non applicable
Support	Local channels lead support.	On the Internet – covers usage in general. First line in bug fixing process.
Training	Competition differentiation.	Includes forms of implementation into existing applications, usage in general

	proprietary software	non-proprietary software
		and ways of developing.
Technology	Ageing.	The one – covered by GPL or GPL compliant license - which suits given enterprise the best. May be changed easily.
Competition	Well established.	No competition or formed.
Market share	Predictable market share every year. Limited opportunities for quick gains.	High growth. Depending on how the project will fill users requirements from the end of this phase market share will drop or slowly rise.
DECLINE PHASE		
Price	High compared to the demand.	Zero or fixed in case of contracted or in house development processes
Promotion	Limited – no promotion or advertising efforts.	Focused on reliability, quality, predictability, availability and usage statistics (covering individual and corporate users). If project has high market share can be presented as stable leader.
Distribution	Use of existing channels.	Through channels known by potential users/customers/developers (the Internet, magazines)
Sales	Maintenance.	Unlimited - product is available on the Internet – may be found on more websites. stable version are published in magazines. In case of contracted/in house project it might be limited.

	proprietary software	non-proprietary software
Development	Focused on cost reduction.	None.
Manufacturing	No capital expenditures, outsourcing.	Non applicable
Support	Phone support.	On the Internet – covers usage in general.
Training	None	Includes forms of implementation into existing applications, usage in general.
Technology	Old and outdated.	The one – covered by GPL or GPL compliant license - which suits given enterprise the best. (Even if is outdated... so what ;))
Competition	Limited.	No competition or formed.
Market share	Shrinking fast.	Depending on project status on the market it can slowly increase or decrease rapidly.

Table 8: Product life cycle phases [First two columns based on product life cycle management] Source: Own elaboration

Because of FOSS quality applications can't be sold as proprietary software is, I would like to make a simple experiment and compare – in general - FOSS and proprietary applications life cycle. The very first phase is project introduction – the longest and the most expensive. Every piece of software has its design, each developer analyses users needs, every application is coded, tested, improved, etc. Of course company, which decide to create its product during traditional – cathedral – development process, has to pay for these activities. Process of creating application can be started because such program is wanted by customer or there is niche on software market or developer or group of them decide to create their own package for fun. The most risky situation is the second ,one when the task is to create 'off-the-shelf' package, which functionality will cover needs of the biggest group of potential users possible at the lowest cost. It is nearly impossible to get to know, what features average user want to use, because of the cost of reaching large group of people interested in piece of non-existent part of

software. During contracted/in-house development process, detailed specification is given by customer. Any change, that has to be made after phase of requirements analysing is finished is the most money and time consuming way. Average cost of changing part of program requiring repeated need analysis is equal to 82 cents for each dollar spent on changing program in overall. In case of FOSS is needless and uncommon to publish final release on the spot GPLed application which contains only main 'engine' and some features is enough. FOSS developers publish their code frequently just to give other people (users, potential developers) chance to work with new application, read code and express their opinion - thanks to that mediocre project has the most wanted features. Such way of working gives solution to problem of coding and testing. Another – more interesting from economical point of view – difference between proprietary and non-proprietary software is fact, that developers are not directly pay by 'customers' (users). When FOSS is developed during contracted developing process only ones who are immediately employed by client are paid directly. Introduction phase is not only a development process, it includes marketing and logistics – in other words sketching overall strategy. Main differences in these parts are a result of 'zero' market and monopoly value of applications, which come covered by GPL or GPL compliant licenses. There is no need to set price – software come for free (except contracted projects). There is no need to care of sales team – it's free, or customer already has bought it – that fact also levels the problem of competition in traditional understanding[56]. Another thing is when the application comes with source code competing is aimless. Joining the project or contributing to it is always more efficient than starting new one. Creating derived works – if the license does approves that – is not a good idea when such derived program differs from the original one only in few aspects. The real use value is a factor taken into consideration when individual or corporate customer makes choice of application to use. Because most projects are found by it's end users on the Internet the key to success on FOSS market is finding the way to inform the most possible group of people that given project exists. Mentioned vortal "sourceforge.net" hosts circa 130'000 of projects and is the most popular (and in fact the best) place, where people look for and find software, which supports their needs.

Of course not everyone will risk and use an early version of given application. During introduction and growth phases program will be used only by those, who exactly know

56 see chapter 5.5 for details.

what they are doing. Usually they are developers and corporate users who have a lot of experience in maintaining such applications. Thanks to their work project reached the maturity phase. The mature phase is for non-proprietary software as important as growth phase for proprietary soft. During this stage free and open source software packages can gain and hold onto market share. Users and potential users are estimating given package technical and economical features. If it turns out, that the new product is reliable, secure, has low TCO and go on the product has a chance to become a leader, what within the FLOSS market means becoming immortal. That is another feature, which differs FOSS projects from proprietary software. Thanks to many iterations of 'development – testing – bug fixing – normal usage – modifications' process developers can create – and users obtain - desirable program, which covers all needed functionality, has no bugs, and due users approval (by using) -became a market leader. Creating another program, which will do same – maybe in different way, but the effect will be the same – is needless and would be completely waste of time. There are many such applications running under GNU/Linux, most of them is responsible for basic system functionality. Because everyone can see exactly how the program works, it is easy to implement it in another[57]. Proprietary software does not give such chance and even the simplest programs are rewritten all over the time.

5.4.1 Alternative marketing forms

When customer can obtain product without charge or for cost of shipment[58], then probably will choose the one, which gives sensation of representing the highest quality. In the world of 'transparent software' the process of making popular package requires only a well coded, useful program with documentation and legible and intelligible source code. This sentence is true even if at the beginning of given project life cycle it covers only small part of functionality desired by end users and even creators. Developers, while choosing projects they will contribute to, concentrate on the values and experience they can gain. Different people devote their time and skills to FOSS projects for different reasons. –As I have mentioned intrinsic and extrinsic motivations of Free and Open Source developers, but users do not really care about anything else, than what they will do profit by using particular piece of software. It is not hard to

57 As a new application understood as external piece of software or simply include its source code in new project..

58 Which is convergent to zero in case of downloading data from the Internet.

estimate if given application is enough secure or stable if it is not already popular, like for example Apache, which is analysed by many individual and institutional users. Proprietary software vendors use traditional methods of attracting new customers to use particular product or group of products. Some companies creates their picture as supporting development in large and small scale by assisting any one – from individuals to whole countries in their way on the path leading to progress. Some want to show themselves as creators of professional and extremely productive solutions for business or user friendly products useful at home or school. Almost every company supports charitable and educational institutions. It is easy to say 'we are nice people, we support those who are in need, our products are good and surely you will make a good use of them' when even a license bans users from publicly expressing their negative opinion on given product. Such prohibition proves that users opinion on program is the most pictorial and renders the real use value of application better than any other description. In the middle nineties Japanese marketing specialists discovered, that the most effective way of product's promotion is so called 'word of mouth'. This technique based on giving young people subjects of promotion - to test or just use it – and some hints how they should praise these products among their friends, colleagues, acquaintances, etc. to encourage them to buy it. It turns out that this simple method is very effective – customers do trust other customers even if there is possibility that person who's advice we are taking is pay and trained to make us to buy particular product. Such trend of promotion is perceptible on the FOSS 'market'. When user can choose one from couple of programs covering the same functionality the chosen application will be the one with the best feedback given by existing users. When program is available free of charge probability, that anybody would pay for positive review or write one discordant with actual state, is convergent to zero. Developers can boast of positive reviews only when their code factually deserve such opinions – they know it and do everything to make code they stand behind as useful as they can do. By having a possibility of verifying what and how professionally implemented features program includes, customer can make clear choice if program is worth interest. Another thing that should be done is to answer to a question if user-customer should use particular product in form in which it is offered by its authors. We can assume, that there is a demand for every new program or at least for a part of it. If an idea occurred to developer there must be at least couple of persons who will find that idea interesting – sometimes such people just require

enlightening that they are in need of newly developed product. If traditional marketing is a process of hiding products fault and bringing into relief its advantages or pretence such existence marketing in GPLed form may be defined as creating product which becomes desirable because of its easily verifiable advantages. Conferring full control of program upon users - by giving them freedom to change, redistribute, create derivatives and etc. - brought to perfection all marketing forms. Product speak for itself, everyone can redistribute program and it's source code and many people do that relieving authors. People just by using non-proprietary software become members of the largest self supporting community. There is no other type of product, which users can boast of being a part of social movement with such intrinsic philosophical background, complex infrastructure, clear and fair rules. Thanks to the fact that cost of product multiplication is convergent to zero, everyone can profit by this 'membership'.

5.4.2 Support

As long as proprietary software is taken into consideration support can be defined as "after sale handholding. Something many software vendors promise but few deliver [...]"[59].

> ***Software Support: Service that software manufacturers, and third-party service companies, offer to customers.***
> ***The Software Engineering Institute- Terms Glossary***

Vendors usually make promises to answer questions about program functionality, questions which mostly begin with 'how', 'why' and 'it is not working'. Most of proprietary software users, who use any form of support, do it because of lack of knowledge, lack of time or lack of proper documentation, which would help to solve their problems. For a company it is less expensive to pay any third party company or to hire somebody with very good knowledge of software packages, which that company is using than to train every employee. Such proceeding save employees time, which results in limiting employer losses directly connected with software unreliability. Following the dictionary definition:

"[...] most support people are useless – because by the time a hacker calls support he

59 From definition in Free Dictionary of Computing.

or she will usually know the software and the relevant manuals better than the support people [...]. A hacker's idea of 'support' is a teete-a-teete with the software's designer".

That is true – many users search the Internet resources, read documentation or learn programs by using but mentioned teete-a-teete is impossible in most cases. Does user knows who is responsible for part of program he or she has problem with... rather not. There are different group of people – those, who create program and those who support its users and that is what makes a huge difference in everyday usage of proprietary and non-proprietary software. It is characteristic for FOSS projects that developers – those who start the project as well as those who only contribute to the project – are in continuous contact with users[60]. Users are interested in getting the most reliable and useful program it can be so they are contributing to the project by using application during many different real life scenarios. Such exploring of application features, most of proprietary software producers have to pay great amounts of money for such testing, leads to improving given piece of software during every phase of its life-cycle. One said that users never know what functionality they would like the new program to cover, they are always unhappy with what they get and it seems that the only cure for that eternal problem is bazaar development style. Existing information exchange channels – board, forums, mailing groups, Usenet, even simple e-mail – give users possibility for describing their needs and experience connected with given application. Freedom to change source code results in frequent patch releases and modifications. Users can became developers and introduce their ideas directly by changing program. Free Software market is one of not numerous markets where informal – based on users & developers community support exists. Before I describe profits that can be obtained by corporate users in the course of support examination I want to recap main features of FOSS support:

- users by reporting questions give feedback about how the program should look like,
- users have actual influence on the program final functionality,
- users are in fact program testers,
- users can become developers and modify programs themselves,

60 They are not only helping users or gathering data about bugs. Users are the best source of information about how the program should look like and what additional functionality it should covers.

- developers gather vital information any software project could be successful without,
- one more time I want to stress importance of fact that FOSS project can be released frequently even if contains untested and buggy code[61].

FOSS users and developers form community within smaller communities, which display is connected to particular projects are formed for the purpose of receiving reciprocal profits. The developers objectives of contributing to Free and Open Source projects, besides the ones described in chapter 4.2.3, are similar to the goals intended by users to gain. Users want to use high quality software and developers wish to create such. The individual user can switch and start using another program more or less easily, but for corporate users such process is much more complicated (that is why continuous[62] support is urgent for them). When a proprietary application producer decides to cease maintaining given software package for any reason (it may become unprofitable due to decreasing demand) or is made to do that (for instance company bankrupt) users are left with binary and documentation alone. If they are lucky management of another company will find it profitable to take over abandoned project along with the existing market and uphold supporting existent group of users. But commercial projects do not die without a reason and such take over is rather improbable. Those who do not want to be held as hostages do switch to FOSS because of source code availability. The most common things companies using non-proprietary software do are:

- use of generally available support channels[63],
- employ coders who fix bugs and/or change program functionality to fit employer needs,
- pay program authors for modifications and to encourage them to work on a particular project.

The first method is the cheapest one, but does not assure that particular problem will be solved in predictable period of time. The number of authors and developers contributing

61 so called unstable versions.
62 continuous means held even after given project is finished and/or founding developers loss interest in it.
63 projects related websites, forums, boards, Usenet groups, etc.

to the project are major factor, which determines mentioned period of expectation. Number of developers depends to a great degree on number of end users – mostly because some users became developers and thanks to the fact that developers are more interested in devoting their time to project which is successful (or at least seems to become one). So it's visible, that the number of contributors should not be the only factor taken into consideration when choosing product with the smallest chance for abandoned. The same fact that a company is going to switch to given application means, that this very package is successful. Of course when the better-known, larger company inform, that their employees will use non-proprietary software, it will attract more developers[64]. Second means to guarantee continuous support is employing programmer. Combining this method with the former one by developing given program in-house and sharing knowledge attained this way with community and at the same time use available resources will not only magnify own productivity but also fasten main developing process. Of course hiring program author or one of main developers is the best solution because such person has the best knowledge on own program and can take the best from other people's contributions. For a huge company adding such people to its payroll is cheaper than these, which it would bear in case of using proprietary packages. From the point of view of small firm' owner ,who keeps busy many fewer employees such disbursal may be comparable to the cost of licenses, paid support and switching to new software in case of mentioned problems. That is why small companies do not establish standards but avail of these constituted by companies, which can afford that. Small companies can simply profit from expenses born by leaders – like smaller animals, which follows an elephant in dense jungle. Problems occurring during everyday program usage in small company will occur or have already occurred if the same program is used by a larger one market competitor. If the problem has not appeared small company can count that it will be solved by a community members much faster than it would be in case of less popular application[65]. All of that brings about situation in which we can charge whole community with solving our problem and it will be done by the first person who find it interesting and achievable. A problem solver does not expect to be paid by us, but there is a possibility that he or she will be paid by our competitor on the market.

64 to see why go back to chapter 4.2.3.
65 advantages of higher market share are described in chapter 5.5.

5.5 *Market share and competition*

Why it is important for users to use application which has significant market share and what menaces it brings follows reasons of product's market share during each life cycle phase[66]?

The developers and two kinds of people who are interested in contributing to new project: programmers and users who need this very application so much that can not wait any longer have real contact with a new piece of soft during the introduction phase. During next phase – growth – program's main features should become stable. The stable version which covers functionality program was destined for is the one, which is a base version for most of users. The moment when developers can supply users with stable version of program[67] and people who only want to use the program – in fact they are becoming very important group of testers - start using such stable version is the end of growth phase and the begging of maturity one. The boost of number of users, which result in growth of developers group, is noticeable during this phase for two main reasons. First one follows the fact that people gain access to useful piece of code, which binary version do things that they need to be done by computer program. In other words: people use program because it is useful. Second is about possibilities which come along with stable version of program – customisation of such application is not biased by main module errors what makes changing it easier and cheaper. Even if somebody does not want to make changes in the program knows that program which already covers main functionality is going to evaluate and become more and more useful with every new version. Maturity phase is the period of time when most of potential corporate users decide to start using given program. Some of them, especially those who are have experience maintaining and using FOSS programs join users/ developers group during first two phases but FOSS newbies[68] and smaller institutional users wait for the moment, when program's popularity guarantee continuous support. David Wheeler wrote, that program's significant market share is lemming-like [wheeler look at the numbers]. The most important advantages of being a market leader is the main reason product become one for – large group of users. Having more customers means greater number of well trained users, greater number of developers willing to

66 see chapter 5.4 for product life cycle walkthrough.
67 version which covers program's base functionality.
68 newbie - any new participant in some activity.

contribute to successful project, lower chance for project abandoned. More interested in given project users/customers means bigger chance for developers for profit. With increasing number of users and developers main factors of package improve but some of them – like security – suffers from programs popularity. I think that when measuring GPLed application[69] market share I should distinguish two different cases:

- The first is based on comparing given application to other GPLed applications which covers the same functionality.
- The second, gives information on position of given application on the market composed of non-proprietary and proprietary software as well.

Because there is no need to multiply existing code, new projects which covers functionality of existing ones, are predestined either to replace their ancestors[70], or become an application with completely different development trend. Sometimes new program replicate the functionality and the source code just because authors want to do everything from the very beginning. It is a common practice that authors include features from different programs as external modules and focus on creating new functionalities. Authors of media player Amarok[71] did it. Amarok is using plugins, which are responsible for decoding media files. While they are developing additional features completely different people are maintaining the program's part of engine responsible for – actually the most important feature of media player – decoding and playing media files. Media players running under GNU\Linux – each is of use to play media files – have such different secondary functionality that they are in fact different programs directed to the different groups of users. At the end of the day competition on the FOSS application market leads to creating products of such different features that makes it interesting for group of users or to death of some products. These commonly used or 'challengers' – or, at the end to the merge of new and existing applications and its developing teams. The only factor, which makes given application popular is its effectiveness, efficiency, supporting team – in overall - application quality so becoming a market leader or part of the same market margin is a simple process. There is no way to persuade anybody into using particular application when there is better one available on the same conditions and switching won't cause additional expenses.

69 or application covered by GPL compliant license.
70 A good example is Apache web server which grew up on the base of NCSA httpd web server. NCSA the project and some of its users decide to maintain it on their own.
71 A project developed by Mark Kretschmann (2002 – 2003) and Amarok Programmers Team (2003 – 2006). More information can be found on Amarok web site (http://amarok.kde.org).

The main technical differences between proprietary and Free/ Open Source software – described in chapter 4.1make FOSS more competitive in some fields and less in others. One of the most successful non-proprietary application on the market is the web server Apache[72] - in fact since 1995 non-proprietary web servers are the most popular[73].

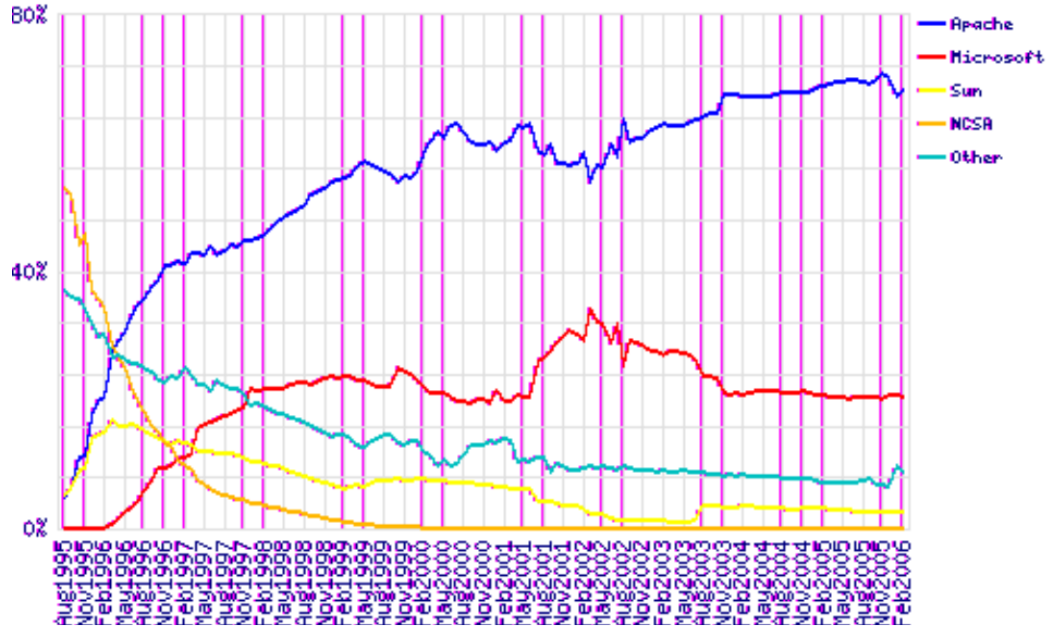

Illustration 12: Market share for top servers across all domains. August 1995 - February 2006
Source: *http://news.netcraft.com/*

It is clearly visible on the graph, that whenever the rapid fall of Apache market share occurred, comparable rise of Microsoft and other companies' products took place. From November 2005 to December 2005 Apache loss 1.01% and the Ms IIS gain 0.68%. We have similar data from different periods (like February 2002 – November 2002). Some Apache market share drops are connected with the expiration of bulk-registered domain names – in December 2005 1 million of hostnames end its' existence at Zipa servers[74] it means that the Apache fits requirements of companies which need fully reliable and productive tool. Decision of registering additional one million of hostnames is strictly connected with forecasts about number of customers which will decide to pay for domain name registered during such promotion – about 80% of such hostnames is going

72 See www.apache.org for details on Apache the web server.
Based on Netcraft's statistics on web servers - http://news.netcraft.com/.

73 The web servers market share can be measured in couple of different ways. Some web site are inactive – domain names are registered but not being used. Web sites can be counted basing on their IP address or their host name. Former way help to remove from statistics computers which hosts multiple sites and sites with multiple names. Some entities also measure number of physical machines.

74 Miller Rich – "Zipa Gains Nearly 1 Million Sites As It Weathers Katrina" ([17]).

to expire, but remaining 20% have to be maintained. Domain registrars are not concerned about Apache reliability or scalability, it seems that web server which is so popular among private users who set it up for fun or for educational purposes can be used to host thousands hostnames and websites. During February 2006 Windows servers gained substantial number of active sites on German and Japanese markets – Intergenia and Excite switch to IIS.

Next graph (illustration 2) illustrates increasing number of sites across all domains, the market is continuously expanding and Apache is a reliable product which create opportunity to set up a business without bearing expenses of proprietary software and licensing.

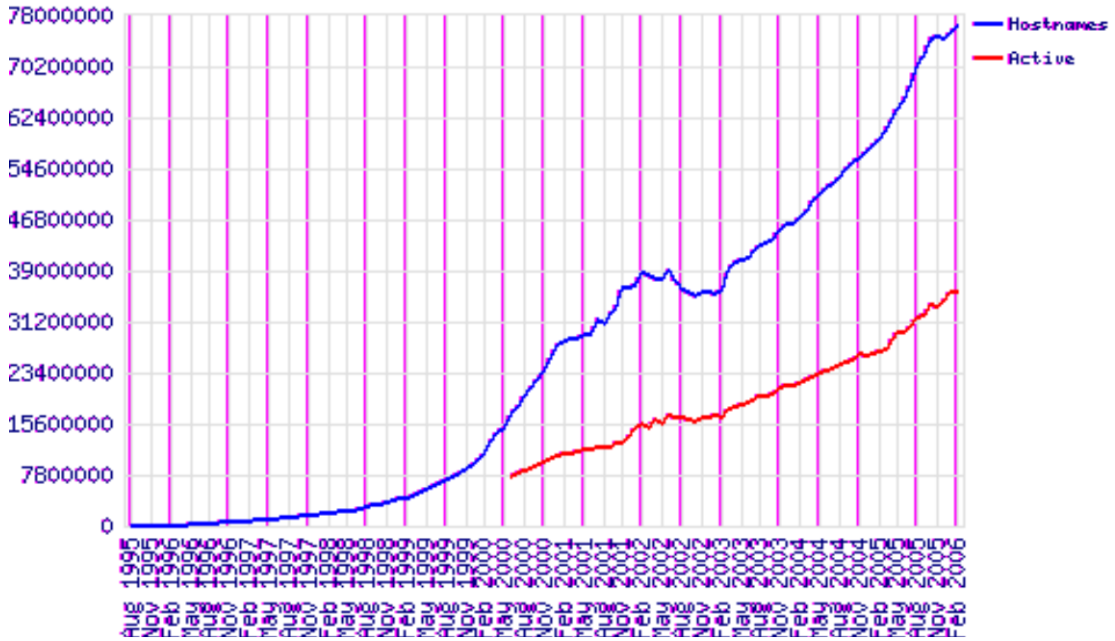

Illustration 13: Total sites across all domains. August 1995 - February 2006
Source www.netcraft.com

On the other hand such dynamically developing fields acquire solutions, which fits the most requirements possible and fact, that Apache has such favourable position on web servers market means, that Free Software Solutions are competitive because of its' advantages.

6 Project development relations – empirical study

6.1 Assumptions

I have been highlighting the importance and impact on software usage of the community's members, who participate in given project development in many ways.

Such people give feedback on the software usage, come with new ideas and implement them. They inform about bugs, or/and fix them. Of course they help other users/ programmers by answering various questions. We can call such mechanisms a support[75]. In this chapter I want to show numbers describing community members cooperation using the example of SourceForge.net projects' groups and forums. The figures I use prove not only that such cooperation exists but that it has impact on projects development process and popularity.

6.2 *Source data and methodology of my study*

All calculations are based on sourceforge.net statistics from database snapshot taken on January 2006. All data presented in this chapter I have received from professor Gregory Madey, who represents faculty of Computer Science & Engineering of University of Notre Dame (Ma, USA). In fact I have gain access to the database, which contains snapshots of most of the tables from sourceforge.net database.

I give careful consideration to relations between projects': completion status, popularity and particular projects' forum traffic. The following table briefly describes used figures.

Figure	description
No. of bugs closed	Number of closed tasks, connected to bugs maintaining, per projects' percentage completion level.
No. of bugs opened	Average number of opened tasks, connected to bugs maintaining, per projects' percentage completion level.
No. of downloads number	Average number of project source/ binary files, per projects' percentage completion level.
No. of help requests	Average number of help requests per projects' percentage completion level.
No. of hours per completion status	Average number of hours which were spent during development process per projects' percentage completion level.

75 More about support one can find in chapter 5.4.1.

Figure	description
No. of patches closed	Average number of closed tasks, connected to process of developing patches, per projects' percentage completion level.
Number of patches opened	Number of opened tasks, connected to process of developing patches, per projects' percentage completion level.
Number of posted messages	Number of messages posted on projects forums, per projects' percentage completion level.
Number of support closed	Number of closed tasks, connected to support threads, per projects' percentage completion level.
Number of support opened	Number of opened tasks, connected to support threads, per projects' percentage completion level.
Number of developers	Number of developers working on particular projects per projects' completion level.
Completion percent	Projects' completion level. Every project is described by one from 21 percentage values (from 0% to 100%).
Number of answered messages	Total number of all messages on all forums which were followed by one or more messages.
Number of forums	Total number of forums.
Number of threads	Total number of all threads on all forums.
Number of threads per group	Average number of threads per project forum.
Number of unanswered messages	Total number of all messages on all forums which were not followed by any message.

Table 9: List of figures
Source: Own elaboration

6.3 *Results*

On January 2006 there was 102124 registered projects, it means that using this service users were able to search through and download source code of 102124 different Free

Software and Open Source Software programs. Projects are divided into 24864 groups. Statistics are available for 90268 projects in 21926 groups.

The relation between completion level and the average number of developers who work

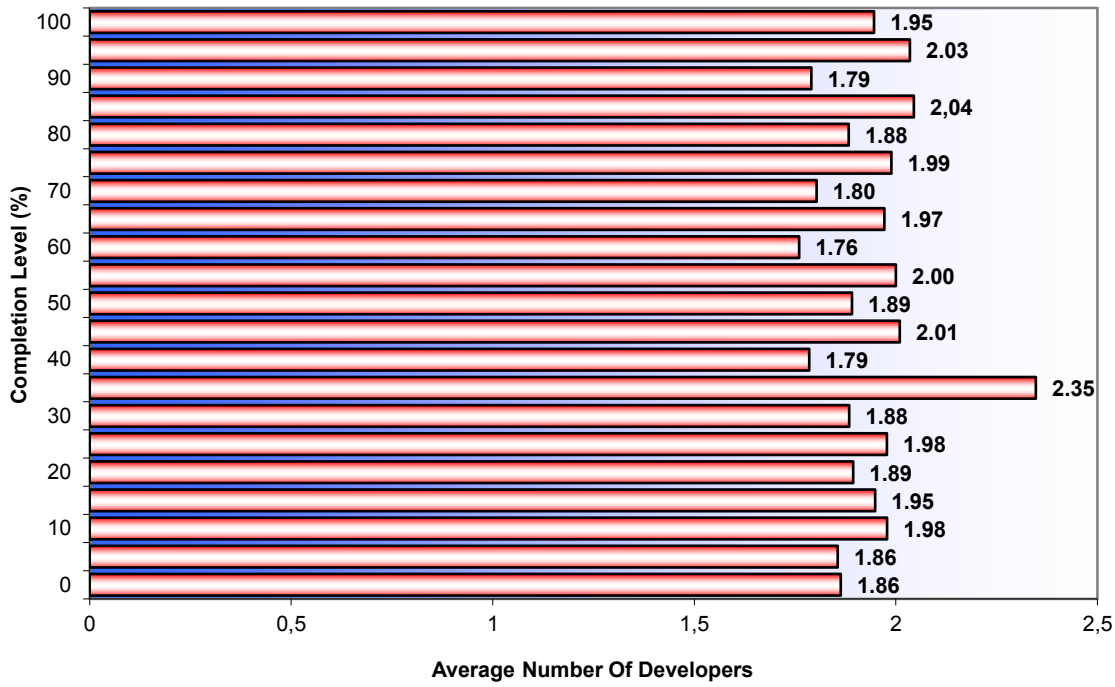

Illustration 14: Relation between projects' completion level and number of developers

Source: own elaboration based on SourceForge statistics from January 2006.

on a project during each level shows that the highest number of developers contribute to projects which are 35% complete. Intensified developers works during this level may be comprehended as the begging of projects' growth phase – period when the most of main program functionality is being implemented. Another interesting thing, which characterise this figure and should be observed, is the initial fall (from 1.98 to 1.89) and fact that from the level of 20% completion average number of developers increase and decrease alternately. It indicates that in the very begging of projects development people who belongs to the group which initiate the project leave project when tasks they were interested to are finished. Afterwards the same or different developers join and leave project dependently on current 'to-do list'[76].

Next very interesting indicator is the average number of downloads which took place during each completion level. The average of 50561 at level of 5% shows the potential

76 todo list – functionalities which will be implemented.

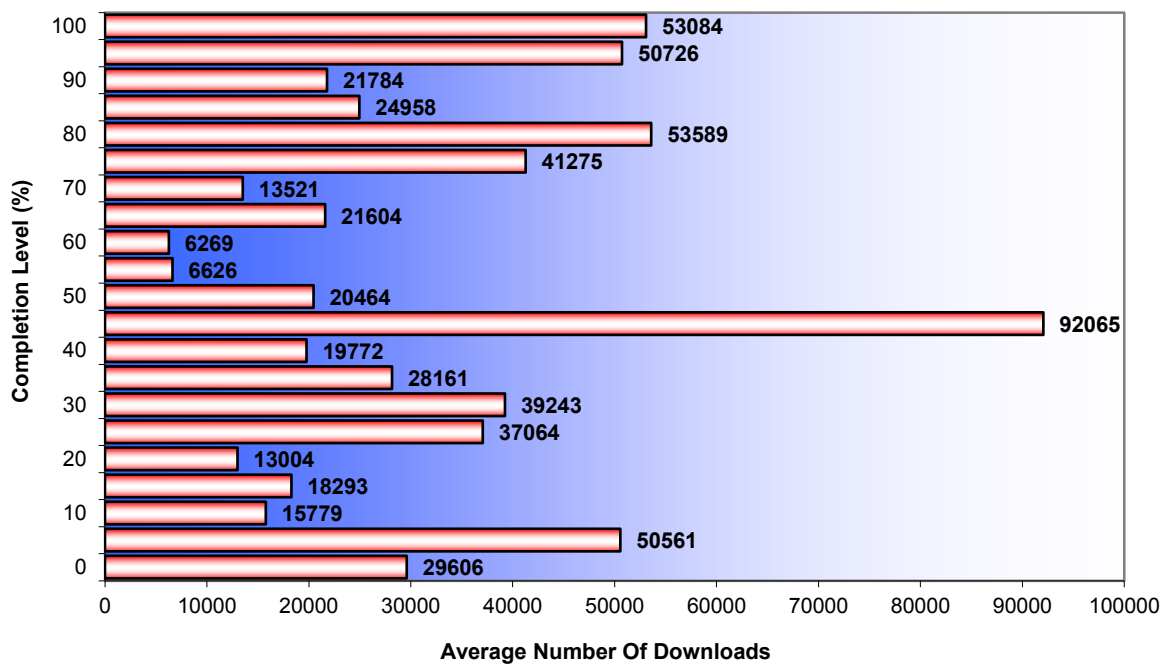

Illustration 15: Relation between projects' completion level and average number of developers

Source: own elaboration based on SourceForge statistics from January 2006.

users and developers interest in newly registered projects. Decreased values at levels from 10% to 20% are the result of boost at 5% and immaturity of projects, which undeveloped functionality does not satisfy users. The second boost can be observed when the project is completed in 30 and 35%. This phenomenon is connected with the beginning of growth phase and increasing number of developers. The levels of 45% draw the end line of growth and begging of maturity phase at 50% of completion. Huge number of average downloads (92065) points that projects' main functions are implemented. From that moment to level of 80% users are waiting for bugs correction and report such during programs usage. The number of projects' tasks connected to bugs maintenance significantly rises on levels of 15, 45, 75, 95 and 100 percent of project completion. First rise precedes growth phase, after which begging at level of 35% is characterised by low number of bugs connected tasks as the result of implementing, testing and correcting primary set of functionality. Second rise is a direct result of introducing to users next part of programs during growth phase which ends with fall of discussed type of tasks to completion level of 55%. Next rise – at 75% - follows bugs reported by users and detected by developers during usage of mature

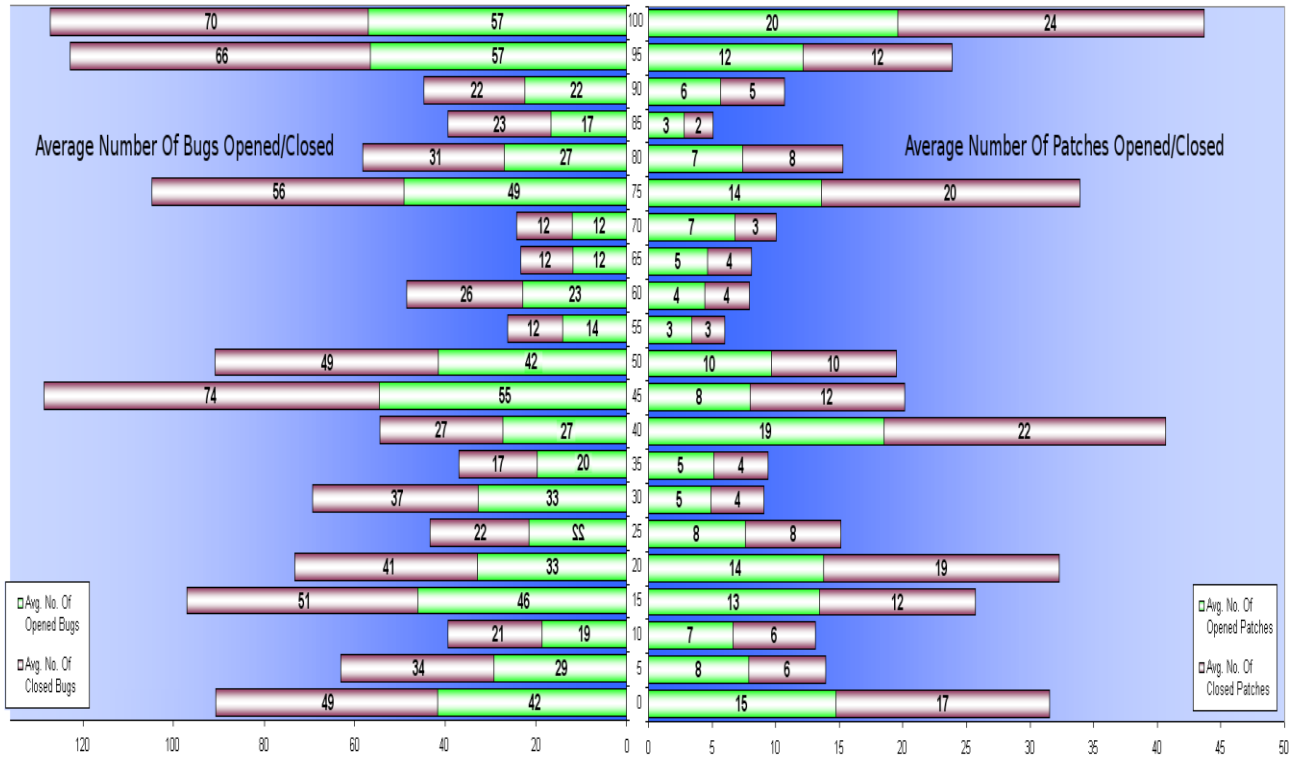

Illustration 16: Relation between projects' completion level and average number of tasks.
green bars – opened tasks purple bars – closed tasks

Source: own elaboration based on SourceForge statistics from January 2006.

versions of programs. This is also the cause of last two rises, during decline phase, when projects are still developed, bugs may be found in code covering new functionality (also number of patches indicates this theory). The relation of average number of patches to bugs points that depending on projects' development phase rises and falls of numbers of this two types of tasks induce themselves in turns or are parallel. For example the rise of bugs' connected tasks at level of 15% induces increase of patches a level later. At the same time number of patches at level of 15% results in series of patches at level of 20%. Bugs detected during growth phase seem to be corrected by patches at level of 40% of project completion. Of course many bugs are corrected in new versions of software and patches and bugs are not strictly related.

The very important feature of FOSS development is the communication aspect. By analysis of the average number of messages sent on projects' forum groups I can draw following conclusions: developers during introduction and growth phase have clearly traced tasks and communication is relatively limited. When projects come into maturity phase number of forum traffic increases mutually with increase of number of

downloads, bugs and patches. In my opinion the traffic boost, which takes place during maturity phase at level of 45% project completion, indicates interest of new users who find project useful. Such traffic may covers help requests, bugs reports, new ideas. I want to remind the enormous rise of number of downloads and fact that average number of developers increases from 1.79 (level of 40%) to 2.01 at this level. Projects completed in 45% are enough reliable and functional to attract people interested in software covering given functionality. In January 2006 in SourceForge forums' archives users could found 752'167 threads. In case of 370'339 threads there was at least on

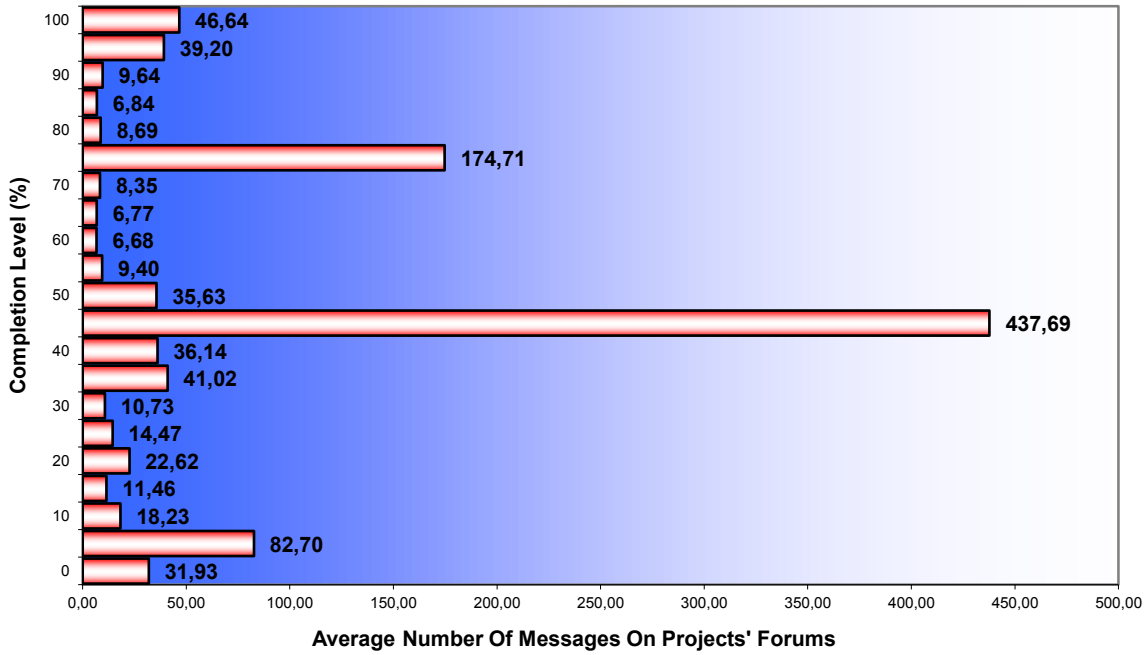

Illustration 17: Relation between projects' completion level and projects' forums' traffic
Source: based on SourceForge statistics from January 2006.

response message – the average was 3.2 message per thread.

In overall – considering all threads average amounts to 1.6. It means that – on an average - every post was answered, or on the other hand that every second post has been answered. Considering that some of threads can be treated as spam (questions already answered in documentation, problems previously reported) I think that such response coefficient can be regarded as satisfactory. Another thing is, that forum threads are addressed to whole community in with hope that there is somebody who knows the

answer or who finds the problem interesting. In terms of proprietary software users, especially companies, which use dedicated software, such way of getting help is nearly impossible. Everybody can freely ask for help or present ideas of needed functionality for free with at least 50% chance for response. The chance for response rises with the project popularity and development level. In terms of using external work sources for own purposes this way of getting help could be a kind of ultra outsourcing.

I want to recapitulate my conclusions by comparing increment of average number of developers, downloads and messages per completion level. The increment of average

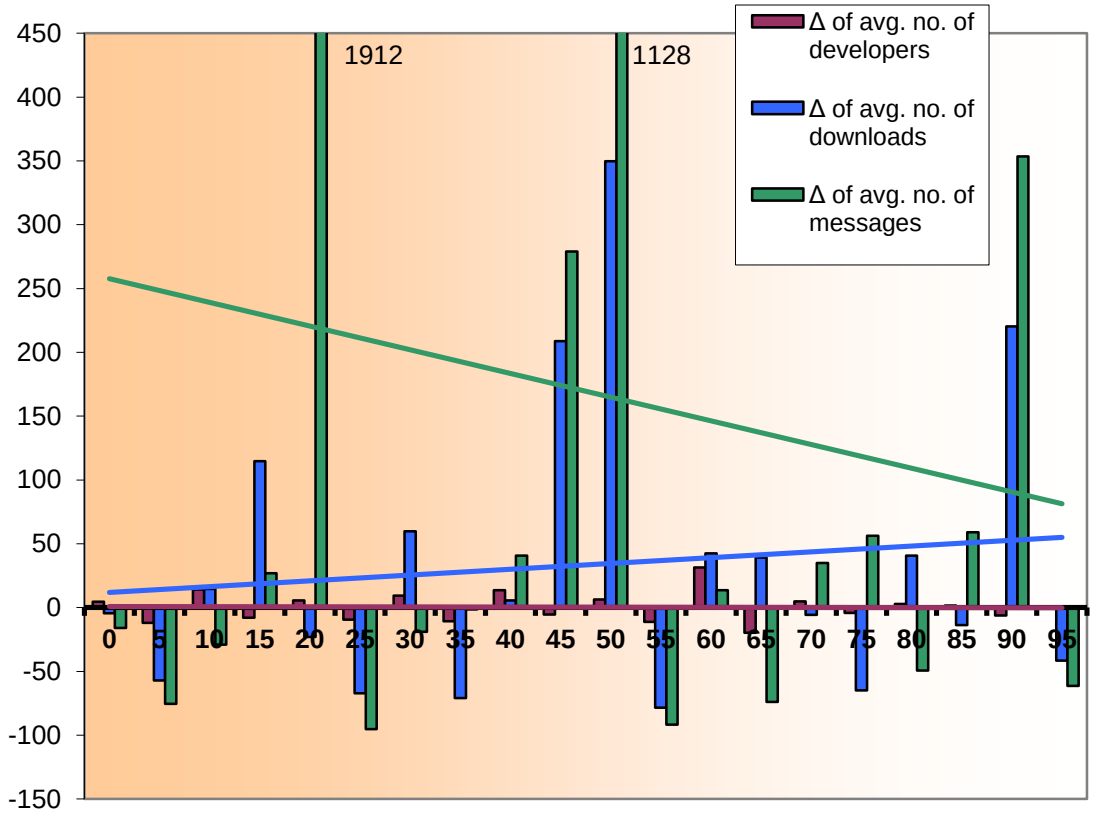

Illustration 18: Increments of avg. no. of developers, downloads and messages (%) per completion level. Source: Own elaboration based on SourceForge statistics from January 2006.

number of developers is in turns positive and negative. Interesting is fact that during begin of growth and maturity phases. The straight line (purple line) representing linear trend of increments of this variable is tangent to axis of abscissas – the overall number of developers during development process can be treated as stabilised. The blue and green lines – which represents linear trends of, in sequence, increment of average number of downloads and messages – indicates that along with project completion level number of downloads slowly rises and number of messages about given project also

rises – especially during growth phase. The significant rise of interest during growth phase – based on the messages traffic per average project along with the number of downloads may results in number of processed bugs and . The software package, which is well developed, and can be treated as useful generate less forum traffic and more downloads.

Life cycle phase	Levels of completion (%)
Introduction	0 – 35
Growth	35 – 50
Maturity	50 – 80
Decline	80 - 100

Table 10: Relation between life cycle phases and levels of completion
Source: Own elaboration

7 The conclusions of the thesis

In my thesis (chapters 3, 4 and 5) I have described my personal approach to some aspects of creating and using Free and Open Source Software. I think that phenomenon of FOSS is so fascinating because of the intuitive and natural way of forming more complicated ways of using it – from philosophy, through law and software life cycle aspects, to technical and logistic solutions. In my opinion the success that FOSS communities have achieved as a result of pointing clear and simple aim – useful and generally available software – was possible thanks to one of basic people's characteristic the desire of freedom. For years many users choose the possibility of having impact on tools they are using for price of nice looking and theoretically ready to work proprietary software. During this years, they have created hundreds of thousands of fully functional, useful, user friendly software packages, which in most cases surpass proprietary ones in case of fundamental, for software products, features like security, reliability and scalability. Simultaneously FOSS community members develop their skills and the culture of knowledge sharing society. In chapter "Project development relations – empirical study" I have proved, above all other things, that the success of Free Software can be measured and described. It means, that it is not a chaotic Utopia. The philosophy introduced by Richard Stallman was transformed into global mental movement, which gives everybody chance for use, create, change and redistribute of software code.

8 Bibliography

1 Boston Consulting Group - "Boston Consulting Group/OSDN Hacker Survey" (2003);

2 Bourke Tony – "Sun Versus Linux: The x86 Smack-down". Osnews.com - http://www.osnews.com/story.php?news_id=4867;

3 Dean Katie - "Data Flood Feeds Need for Speed" – Wired News, 13.02.2003 - http://www.wired.com/news/infostructure/0,1377,57625,00.html

4 Davis, A.M. – "Software Requirements: Analysis and Specification" - Prentice-Hall, 1990

5 DiBona Chris, Ockman Sam, Stone Mark - "Open Sources - Voices from the Open Source Revolution"

6 Free Software Foundation - "The Free Software Definition" - http://www.gnu.org/philosophy/free-sw.html

7 Free Software Foundation - "Why 'Free Software' is better than 'Open Source' " - http://www.gnu.org/philosophy/free-software-for-freedom.html

8 Free Software Foundation - "Overview of the GNU Project" - http://www.gnu.org/gnu/gnu-history.html

9 Free Software Foundation – "Free Software Foundation" - http://www.gnu.org/fsf/fsf.html

10 Free Software Foundation - "GNU Public License" - http://www.gnu.org/licenses/gpl.html

11 Haruvy, Wu, Chakravarty - "Incentives for Developers' Contributions and Product Performance Metrics in Open Source Development: An Empirical Investigation" - University of Texas, Dallas, 2003

12 Howorth Roger - "Samba 3 extends lead over Win 2003 Roger Howorth" - IT Week 14 Oct 2003 - http://www.itweek.co.uk/News/1144312

13 IBM - "Putting Linux reliability to the test" - http://www-106.ibm.com/developerworks/linux/library/l-rel/

14 Kotonya, G., Sommerville, I. – "Requirements Engineering: Processes and Techniques" - John Wiley and Sons, Inc, New York, 1998

15 Lakhani, von Hippel - "How Open Source Software Works: 'Free' User-to-User Assistance" - 2003 - Research Policy

16 Lerner, J., Tirole, J. - "The Economics Of Technology Sharing: Open Source And Beyond" - Working Paper no. 10956 – National Bureau of Economic Research, December 2004 - http://www.nber.org/papers/w10956

17 Miller Rich – "Zipa Gains Nearly 1 Million Sites As It Weathers Katrina" - Posted on Oct 26, 2005 - http://news.netcraft.com/archives/2005/10/26/zipa_gains_nearly_1_million_sites_as_it_weathers_katrina.html

18 netcraft.com – "Hosting Providers sites ordered by failures" - http://uptime.netcraft.com/perf/reports/Hosters

19 netcraft.com - "Hostway most reliable hoster in November" – 2005 - http://news.netcraft.com/archives/2005/12/14/hostway_most_reliable_hoster_in_november.html

20 Netscape Communications Corporation – Press Release, 22.01.1998 - "Netscape Announces Plans To Make Next-Generation Communicator Source Code A Viable...." - http://wp.netscape.com/newsref/pr/newsrelease558.html

21 Office of Technology and Electronic Commerce (OTEC) division of the International Trade Administration, U.S. Department of Commerce - Size of the U.S. Computer Software Industry – 14.04.2003 - http://web.ita.doc.gov/ITI%5CitiHome.nsf/AutonomyView/87200518f179196c85256cc40077ede1

22 Open For Business Project, The - "Philosophy" - http://www.ofbiz.org/philosophy.html

23 Open Source Initiative - "Why 'Free' Software is too Ambiguous" - http://www.opensource.org/advocacy/free-notfree.php

24 Perens Bruce - "The emerging economic paradigm of Open Source" – http://perens.com/Articles/Economic.html;

25 Rossi C. and Bonacorsi A. - "Intrinsic motivations and profit-oriented firms in Open Source software. Do firms practise what they preach?" – (2003)

26 Samoladas Ioannis, Stamelos Ioannis, Angelis Lefteris and Oikonomou Apostolos - "Open Source Software Development Should Strive for Even Greater Code Maintainability" - 'Communications of the ACM' – October 2004, pp. 83-87

27 Stallman Richard - "The Initial Announcement" -

http://www.gnu.org/gnu/initial-announcement.html

28 Stallman Richard - "The GNU manifesto" - http://www.gnu.org/gnu/manifesto.html

29 Symantec Corporation - "Symantec Internet Security Threat Report – Trends for July 05 – December 05 – Volume IX"

30 Wheeler David - "Open Source Software / Free Software (OSS/FS) References" - http://www.dwheeler.com/oss_fs_refs.html

31 "Reasoning Releases New Study Showing Open Source Model Produces High Quality Software" - http://www.businesswire.com/cgi-bin/f_headline.cgi?bw.021103/230420300

9 Appendixes

9.1 The list of terms and abbreviations

- **Availability** degree to which a system suffers degradation or interruption in its service to the customer as a consequence of failures of one or more of its parts;
- **Beowulf** multi-computer architecture which can be used for parallel computations. Frequently composed of one tie-server;
- **Copyleft** the minor feature of copylefting is to grant rights "...to use, modify and redistribute the programs' code or any program derived from it but only if the distribution terms are unchanged";
- **Cracker** an individual who attempts to gain unauthorised access to a computer system. The term was coined ca. 1985 by hackers in defence against journalistic misuse of "hacker". (from definition in 'Jargon file' by Eric S. Raymond).
- **EMACS** the extensible, customizable, self-documenting real-time display (text) editor**;**
- **F/LOSS** Free/Libre Open Source Software;
- **FAQ** Frequently Asked Questions;
- **FOSS** Free Open Source Software;
- **FS** Free Software;
- **FSF** Free Software Foundation;

- **GNU** the recursive acronym for GNU is Not Unix;
- **GPL** General Public License (aka. GNU GPL);
- **GPLed** covered by GNU GPL;
- **Hacker** One who programs enthusiastically (even obsessively) or who enjoys programming rather than just theorizing about programming;
- **KDE** K Desktop Environment;
- **LGPL** Lesser General Public License (aka. GNU LGPL);
- **License** an authority or liberty given to do or forbear any act, especially, a formal permission from the proper authorities to perform certain acts which without such permission would be illegal;
- **Newbie** any new participant in some activity;
- **OOP** Object Oriented Programming;
- **OSI** Open Source Initiative;
- **OSL** Open Source License;
- **OSS** Open Source Software;
- **Performance** The way in which a machine or other thing performs or functions: behavior, functioning, operation, reaction, working;
- **QTPL** Qt Public License;
- **RAS** Reliability, Availability and Serviceability are features regarded as ones, which describe software quality in the widest way;
- **Scalability** the ease with which a system or component can be modified to fit the problem area;
- **Security** the ability of a system to manage, protect, and distribute sensitive information;
- **SLOC** Source Lines Of Code;
- **Support** Service that software manufacturers, and third-party service companies, offer to customers.

9.2 List of figures