Ph.d. Outline

This is a working paper and may not be cited without permission. Please mail any suggestions or comments to ke@its.dtu.dk. References used in other chapters have been included to provide an overview of literature used.

Kasper Edwards, Lyngby the 3. April 2000

This chapter gives an introduction to the thesis as a whole. The history of computer operating systems and a brief description of the development of software are presented along with a description of a computer program. Subsequent, the interesting aspect of developing software in the GNU/Linux community is presented.

Following the introduction the theoretical motivation is presented. This paragraph describes the theoretical areas that have inspired this thesis. Later the research questions are presented.

1.1 Contents

1.1 CONTENTS 1
1.1 EMPIRICAL INSPIRATION 1
1.1.1 What is interesting about the GNU/Linux operting system? *3*
1.1.2 Is it just GNU/Linux? *4*
1.1.1 A word on developing software *5*
1.1.2 Different types of programs *6*
1.1.3 A word on how software are developed in the GNU/Linux community *7*
1.1.4 Keeping track of version numbers *8*
1.2 THEORETICAL MOTIVATION 10
1.2.1 A development model *10*
1.2 RESEARCH QUESTIONS 11
1.2.1 What characterises the GNU/Linux development model? *12*
1.2.2 What are the economic mechanisms of the development model? *13*
1.2.3 What are the implications of using the GNU/Linux development model *13*
1.2.4 Does the GNU/Linux development model apply to other than software? *13*
1.3 EXPECTATIONS 13
1.4 BOUNDARIES OF THE REPORT 14
1.3 THE STRUCTURE OF THE REPORT 14
1.5 REFERENCES 14

1.1 Empirical inspiration

Proprietary operating systems have through the history of computing been dominating the scene of operating systems. There are examples of free operating systems that have been released to the public, one of the first one was the version 1 of AT&T Unix from 1971. Later when the licenses from AT&T got more restrictive and expensive there was a reaction from the Unix user community – the release of 3BSD the 3'rd release of the Berkeley Software Distribution. This was a free Unix operating system, later came NetBSD, FreeBSD and OpenBSD all free Unix operating systems. Parallel

to this development many commercial vendors entered the lucrative market for operating systems and began developing their own proprietary versions of Unix. Some of the Unix'es were compatible and others were not. All these Unix'es had one thing in common, they were made for the very expensive hardware that existed in those days. This kept Unix in the hands of corporations and universities and kept it safe from the private user. This also meant that potential private developer did not have the possibility to contribute. The different Unix'es have always been competing for users and customers and all the time have the mentioned in-compatibility been existing. This made it expensive for software developers to create software for all types of Unix. It is time consuming and costly to maintain a piece of software for all types of Unix and many vendors chose to market for only a select number of Unix'es. This might be the reason why Unix in the past never became the most dominating operating system and lost market to Microsoft's operating systems. These may have been technically inferior but there was just one operating system for which to write applications. The developers and users could just choose the Microsoft operating systems and gain simplicity in development and a wide selection in applications.

The large majority of Microsoft users were from the beginning users that bought a computer to solve a specific purpose like word processing, spreadsheet or other needs. Only a fragment of these users was interested in computer science. These users were mostly interested in solving a problem and making it easier to work – not developing the system. The companies that evolved around the PC were part of this new type of computing not focusing on the user as a source of continued development and thus not supplying the source code.

It was not until the advent of the Intel 80386 microprocessor for personal computers in the late 1980'ies that Unix for the PC became interesting. The 80386 had some of the advanced features needed to run a Unix operating system. The 80386 introduced a 32 bit architecture, memory management and others features that made the PC a cheap alternative for using Unix.

In 1991 the finish student Linus Torvalds was fed up with the operating system on his Intel 80386 (a particular type of CPU, central part of a computer) PC-computer. Wanting a more power full computing environment Linus Torvalds began to create his own Unix like operating system, which he named Linux. Linux was released to the public late 1991. Linus Torvalds created Linux using a set of development tools (compiler, linker, debugger and other) developed by the Free Software Foundation. These tools were free and were cover by the GNU Public License. The GNU Public License[1] (GPL) allows the user to change and modify the code. Even redistribution is allowed as long as the changes are made public and derivative work must carry the GPL license.

The Free Software Foundation was created in early 1980'ies as an effort to create and distribute free software. Linux is actually only the kernel (the central part of the operating system). It is hard to imagine the birth of Linux without the development tools form the Free Software Foundation. It therefor seems reasonable to call the

[1] GPL licensed programs grant the user permission to run, copy, distribute and modify the program. It is not permitted to add restriction and thereby impose propriety rights on the code. Any GPL code used in a program makes the program GPL. Once a program is GPL'ed it will remain GPL. [Stallman 1999:60].

operating system, the GNU/Linux operating system or just GNU/Linux depending on context this reflects the origination of the operating system as a whole. GNU is an recursive acronym for Gnu Not Unix, this is reflecting on the fact that the word Unix is trademarked by AT&T and can not be used without AT&Ts permission.

1.1.1 What is interesting about the GNU/Linux operting system?

The GNU/Linux operating system in it self is not interesting in relation to this project. The operating system and the related software becomes interesting when viewed as a product of a particular development process. The development process becomes interesting because of the following factors:

- The developers are working for free.
- The resulting product is free.
- The license allows everyone to elaborate and use parts or the whole operating system in their own project.

These three factors represent a glaring contrast to the commercial way of developing software that have resulted in the operating systems that are used in most computers around the word today. Examples of commercial operating systems are the Microsoft Windows series of operating systems, Novell Netware, IBM operating systems for their mainframe computers and some of the proprietary Unix operating systems. All these operating systems have been developed by paid workers earning their living as programmers. The resulting product from the development has been operating systems – products – meant to provide the developing company with a profit ensuring the continued survival. Never has the product been free, although development versions have sometimes been free. The licenses used have always stated that the user was not allowed to change of modify the product and that the technology used in the product were property of the company. This made sure that no one had legal rights to continue the development of the products. The companies of course never released the source code for their products without a signed nondisclosure agreement.

Within the last few years the GNU/Linux operating system have been experiencing a tremendous following with an exponential growth in installed base. On server side the GNU/Linux operating system is the system of choice among web server the GNU/Linux operating system are claming a market share in excess of 50%. Linux is experiencing a growing support from vendor of server and database products being one of the prime examples are Lotus, Informix and Oracle has released products. IBM is offering support for GN/Linux for their line of server hardware configured with GNU/Linux.

The client side lacks the same momentum, though it is the Unix system of choice for personal use. Still compared to the number of Windows installations in use it holds but a fragment of the market. The client-side is, however, experiencing a major growth in software support from commercial vendors. WordPerfect Corporation has recently released a full Linux distribution intended to be the easy-installation product the non-tech user this distribution is of course including the famous WordPerfect word processor. Games are also starting to be released for the GNU/Linux operating system, this has until now been a non-existing realm of software for the GNU/Linux community. These are all indications that the GNU/Linux installed base will continue to grow.

The support from commercial vendors is an indication that corporations believe that money can be made and that Linux may be viewed as an emerging market. The corporations contribute to the development of Linux but it must be remembered that unpaid individuals developed Linux, the GNU tools and much of the software that are freely available. This still holds true at the time of writing the majority of software are being developed and maintained by unpaid people. Some programs have been developed by a single programmer other have been developed by a group of programmers collaborating.

The continued growth of GNU/Linux relies not on a single software vendor but on the collaborative effort of the GNU/Linux community. The GPL license applied to GNU/Linux prevents a single person or company from securing the rights for the system and gain control of the development. It also ensures that effort made on existing parts of GNU/Linux will be returned to the community.

The GNU/Linux operating system thus seem to be developed in a special way that allow all kinds of developers to participate, private a well as corporate, in a process where every body has some control and no one has complete control. The products of development are free software that has a potential profit attached in terms of use and sale. The free (gratis) nature of the software makes it is difficult to understand the incentives for participating in development of software.

The importance of understanding this way of developing software is emphasised by the success of the GNU/Linux operating system. People form all parts of the world have participated in developing a full-featured operating system capable of competing with major payers in selected areas e.g. web and file/print servers. The interest shown by commercial companies and private home users contribute to the understanding of GNU/Linux as a success. The size and growth of companies engaged in GNU/Linux activities, the growing user base and the longevity of GNU/Linux makes it clear that GNU/Linux is not a merely a curiosity. GNU/Linux is here to stay and will continue to develop. The task at hand is to understand the way it is developed.

1.1.2 Is it just GNU/Linux?

So far only the GNU/Linux operating system has been mentioned this is mainly due to the fact that Linux have been a high profile name in the press and the profile is still growing. Linux and the notion of a free operating system and free software have reached the political system with some parties demanding government software to be based on free software[2]. The French government has decreed that within three years a certain amount of software must be based on free software. A factor in these decisions may be related to a single company supplying a vast majority of the software and that company resides in a foreign country.

The high profile of GNU/Linux cast a shadow on the large amount of software that are not produced by neither Linux kernel developers nor the developers related to the Free Software Foundation. Actually GNU/Linux is but a part of a much larger

[2] "Information" 14. February 2000 p. 6.

community developing free software. When analysing[3] the number of people mentioned in the credits file the number reaches 286. One might think that this is a large number but many others have contributed to the vast number of other software projects the fact is that there are many projects going on and many people participate in these projects. A quick scan of projects mentioned at http://www.linux.org lists more that 120 projects. Some projects have less than 10 active developers others have more than 250 active developers. The projects have different perspectives some are information projects that produces different kinds of documentation others are projects that tries to get Linux running on a different type of computer. Some projects develop high performance specialities such as multiprocessor computers and distributed computing. The there are a large group of software projects developing windowing managers, a flight simulator, 2D mechanical cad systems - in short, all kinds of software.

GNU/Linux it self gets all the credit but the quick search at linux.org showed that many many other people are engaged in developing software and others related to the GNU/Linux operating system. Add to this that linux.org doesn't even begin to cover all the projects that are in active development. Many projects are only announced within a certain User Group as one example SSLUG (Skåne Sjælland Linux User Group) have currently four projects[4] that are only announced on the SSLUG homepage. From web sites such as freshmeat.org who's mission is to convey information on new projects and the status of existing projects it is apparent that there are a vast amount of software that are developed in a similar manner as the GNU/Linux operating system. That is: Developed by unpaid programmers, the resulting product is free and the license allows people elaborate on the programs.

1.1.1 A word on developing software

The results of the collaborative effort of people in the GNU/Linux community are software, a program that can be used on a computer. Software does not spontaneously begin to exist but are developed and refined over time by inspired developers.

Programming or coding, the preferred term by hackers is the art of writing a sequence of instructions that a computer may execute and thus perform the tasks wanted. The creation of a program may be described as a blend between rigid syntactic perfection and the creativity of building a castle out of nothing. Each and every command has to be stated according to correct usage any misspellings or wrong semantics will not be understood by a compiler. Coding gives the programmer absolute control, anything can be created, the programmer just has to know what he want then there are infinite different ways the program may be coded. Of course there are good and bad ways to implement a program depending on the given situations.

The actual task of writing of a program is done the same way a text document is typed and edited in a word processor. Instructions are entered, edited and the resulting

[3] When using Linux the number can be counted by issuing the following command: 'grep \^N: /usr/src/linux/CREDITS | wc –l'. Thanks to Henrik Christian Grove, news.sslug.dk sslug.misc:7206 for pointing this out.

[4] The projects are Danish/Swedish spelling, the Danish Howto, cvs2html and Localising (conversion to Danish and Swedish standards of currency, metrics etc.).

source code is saved in a file and compiled into a program that can be executed on a computer. A compiler is a piece of software that translates a source code into something a computer can understand and execute. There exist different kind of programming languages that a programmer may use. The languages have different features that make some languages better suited to solve some problems rather than others. The development of the GNU/Linux operating system and much of the related software are done in the language C. The C programming language was a spin-off from the development of Unix at AT&T in the beginning of the 1970'ies. C is used because it is standardised[5] by IEEE and ISO which means that a program written in C are likely to run on many operating system and hardware. C is also noted to be an abstraction just above the hardware layer meaning that the programmer has much control of the hardware but is still accessing the hardware through a language abstraction. C is praised to be simple but yet powerful sometimes the word beautiful even comes up.

1.1.2 Different types of programs

Programs are not just programs some programs are large some are small even others are very complex. Here, to clarify a bit, programs are divided into different types, in "The Mythical Man-Month", Fred Brooks outlines four types of programs that seem sufficient to ease understanding of programming and different programs:

1. A little program
2. A Programming product (generalisation, testing, documentation)
3. A Programming System, (Interfaces and system integration)
4. A programming systems product, (Combination of 2 and 3)

A little program is an entity in it self, it can be compiled and run on a standard system. The little program is the creation of the individual, little comments and remarks are made in the source code. Understandings of the program design are retained in the mind of the programmer and there is a lack of explicit documentation. Often this implies that the structure of the program is less clear and that the programmer has no defined coding strategy. Elaborating on the program by other than the author is severely hampered hereby. Programming is however stimulated by the apparent progress as many lines of code are written without wasting time on structure and documentation. More often than not the little program solves a specific problem and is not designed with general usage in mind.

The programming product is a program that has been extended in such a way that any programmer can maintain the program by making changes in the code. The program is now more structured and commented so that other programmers understand the design and choice of specific programming technique in different parts of the program. The usability of the program has been enhanced into something that can be used to solve several types of problems. The programming product is tested in different contexts i.e. in conjunction with other programs and under various

[5] C was developed to ease the porting of Unix in 1972. Many versions of C have been developed but in 1986 a committee under IEEE was commissioned to define a standard for the C programming language. The second version of the C standard have been released resent??? Ref og årstal mangler.

configurations. Testing is very time consuming and requires substantial resources. Still the programming product is a single program that is not dependent on other programs to perform its function. Along with the commented source code the programming product is documented. Estimated project time is 3 times that of a little program with a similar amount of instructions.

A programming system differs from the above by using several different programs to perform its function. It is a system of programs that interact and work together to perform its tasks. The programming system appears, to the user, to be a single program though it is many little programs are performing different tasks. Often the programming system is the result of several people working together and the different tasks have been delegated as individual programs. The challenge for the programming system is requirement for the individual programs to conform to standard for input and output. For data to pass between individual programs components have to have a standard describing the interface between components. This has to be done in detail defining details such as interface, syntax and semantics. Co-ordination between involved parties is often a problem in the programming system where initiative and progress may cause incompatibility between the individual programs in the system. Testing is complicated by the number of programming components in the system and the amount of errors rise with the combination of programming components. Estimated project time is 3 times that of a little program with a similar amount of instructions.

A programming systems product have the all the characteristics of the above. It is programming product where structured programming has been applied and good testing and documentation have been conducted. In addition it is a programming system with many interacting individual programs where elaborate testing have been conducted. Estimated project time is 9 times that of a little program with a similar amount of instructions. The estimated time is derived from the combination of a programming product (3 times) and a programming system (3 times).

1.1.3 A word on how software are developed in the GNU/Linux community

The majority of programming projects that is started in the GNU/Linux community is initiated by an individual person creating the first version of the program by himself. Not working with others limits the required infrastructure to a computer with the GNU/Linux operating system installed which includes all the tools needed for developing software.

When a program reach a state where it becomes usable to other, the community reacts by beginning to download and try the program. If the program meets expectations and/or requirements people begin to participate in the development by submitting comments, bug-reports, suggestions for improving the program or even patches (a patch is an update to the code from the program a patch can replace and/or add lines to the code). This downloading and communicating between interested parties requires more infrastructures compared to the development of a little program by a single person. In essences the infrastructure has to support communication between the parties that participate in the development process. Two types of communication takes place: 1) Information regarding the project, what direction the project is going, discussions regarding coding and general discussions. 2) Program and updates, this is code fragments, patches and new versions.

The non-code communication can be done using one or more of the internet based forms of communication. Web pages are commonly used for general one-way communication from the maintainer of the project to all interested parties. A web page provides the basic information regarding the project and describes how to communicate with the project. Two-way off-line communications are done using email, mailing lists, newsgroups and related services. One-line communications between interested parties are done using services like IRC, chat lines and others. IRC (Internet Relay Chat) is an internet based chat forum where people can enter a specific channel to chat online, IRC allows creation of private and group channels. Depending on the project one or more of these forms of communication may be employed.

Code communication is the distribution of project related code between the interested parties. The most common form of distribution is ftp-download together with a web page, many internet sites provide these kinds of services for free examples are www.freshmeat.com and www.sourceforge.com. Code may also be distributed by email and mailing lists.

Maintaining a project with a small number of people participating can be done by hand of the maintainer. When several people contribute to the project the administrative burden becomes large and the use of automated features becomes appealing. Projects with automated procedures for updating the program are often centred around a server running a program like CVS (Concurrent Versioning System) other programs exist but the core functionality remains the same. CVS is a program that track changes in program file or a number of files every change receives a number and is added to the version tree as an independent file. It is possible for the programmer to download select versions or just recent changes form the server. The brilliant thing is the ability to separate the versions. A person may request version x.y.z. and the server will assemble the specific version for download. Thus it is possible to test things and go back to previous versions when a change breaks the program.

1.1.4 Keeping track of version numbers

A mantra in the GNU/Linux community is “Release early, release often” meaning that when changes have been made or features added don’t wait until extensive testing have been done release to the community without hesitation. Of course many of the new versions introduces bugs along with new features and improvements. This calls for a versioning system that, at a glance, can inform of the status of the program.

Releasing a program without extensive testing is called releasing a development version. A development version that have been tested in the community and received feedback may later, when bugs have been documented and remedied, be released as a stable version.

The GNU/Linux community applies a de facto[6] version naming scheme. The version naming scheme uses tree ciphers to convey the information e.g. Linux kernel x.y.z. The x is the major version number, the y is the sub-version number and the z is a revision number. Some distributors like Red Hat applies a fourth number e.g. x.y.x-cc

[6] A de facto standard is the standard in use that the majority conform to. De facto is opposed to a de jure (formally approved) standard.

where the cc refers to a distributor specific version number. The version system in the GNU/Linux community differs from the normal software version numbering with the addition of the third and sometimes fourth number. There is also a conceptual difference, the GNU/Linux community use of versions reflects that versions have great importance in the every day use and that users are treated as developers. A use can easily submit a bug report and with little effort identify the specific version. This is as opposed to the versioning system of commercial software like Microsoft Word or WordPerfect that actually carries two version numbers: An official number like Word97 and a technical version number like 53360-270-5502054-62850 that is the reference to the actual code used in this specific version. The official version number have a much greater PR value that technical like when MS Word changed version number from 2.0c to 6.0 to be able to compete with WordPerfect version 6.0.

The major version numbers, in the GNU/Linux community, refers to major change in the program. The change from version 0.y.z to version 1.y.z usually means that a program have gone from early development to usage stage. Historically the version 0.y.z was internal development versions and version 1.y.x became the first official public release. Further changes in the x number signals some major change in the system, an example from the Linux kernel development, when the version shifted to version 2.y.z, a new file system was introduced and kernels with lower number were incompatible with the new file system.

The second number, y denotes two things: Equal number indicates that the version is stable and tested and should be used for production. If the number is unequal it indicates an experimental version that are still in development. Following development and testing an unequal version may be promoted to a stable version with an equal number.

The z number indicates the revision number. Revision numbers rise as new versions are developed and made public. At one point in time the specific version is locked in terms of wanted changes and the version is tested until approved and then released as a stable revision with an equal number.

1.1.2.1 Who decides the numbers

Historically it has been the maintainer of a project that has decided when enough changes had been made to make it a new version. To a large degree it is a matter of bounded rationality, when a new version is released as a development version people starts to use it and bugs and missing features are reported to the maintainer of the project. The list of bugs and missing features begins to grow and at some point the maintainer decides to close to list for new items. Now a period of time passes where a suitable amount of entries from the list gets done and a new release is ready. Depending on the status of the program the new version may be released as either a development or a stable version.

The version naming scheme are being used by most of the development projects. The process of developing and testing varies, however, in the different projects depending on different factors. The factors include the history of the project i.e. the experience gathered in the project, available resources, man time, testers, users and others.

1.2 Theoretical motivation

The one motivating theoretical motivating factor is the apparent lack of theory that is able to explain evolution of the GNU/Linux operating system and related software. Remembering that software developed in the GNU/Linux community is developed by unpaid programmers, the resulting product is free (gratis) and the license allows people elaborate on the programs there are some theoretical problems.

Theory on the economics of innovation has grave difficulties understanding the development of the GNU/Linux processes. Going back to the early work of Josef Schumpeter often referred to as Schumpeter Mark I the conceptual problems of understanding the development of GNU/Linux is noted already in the premise. In Schumpeter Mark I the central figure is the entrepreneur, a person motivated to start up a new business, who is driven by the expectation of a temporary super normal monopoly profit. The entrepreneur expects to be first to market and there by become a monopolist for a short period of time. But by the status of a monopoly the entrepreneur will be in a position to charge extra high prices. The entrepreneur becomes an innovator by introducing new technology to the market.

In Schumpeter Mark I the premise that prompts the entrepreneur to act is the expectation of a monopoly profit. I one views GNU/Linus as a result of the innovative effort of entrepreneurs one would have to find the profit incentive that have motivated the entrepreneur(s). Staying within the Schumpeterian tradition this would mean that GNU/Linux should be able to generate a monopoly situation and thereby a super normal profit that would reward the risk taken by the entrepreneur(s). The license of GNU/Linux and much of the software are, however, explicitly stating that the software may be distributed and used in other software. This is in effect the exact opposite of a monopoly where the entrepreneur retains all rights. Thus, the Shumpeter Mark I theory of innovation already at the premise is unable to understand the development of the GNU/Linux operating system and software that carries similar licenses.

Many of the theoretical problems of understanding the development of GNU/Linux and related software stem from the missing incentives of why the agents act the way they do.

The one of the premises of the neoclassical economic theory is that agents maximise their profit this is understood in a monetary sense. In the development of GNU/Linux agents act, produce and innovate but apparently not to maximise a monetary profit but for other reasons or incentives.

1.2.1 A development model

It seems appropriate define the way GNU/Linux and related software are developed as a development model. This emphasises that this is a certain way of developing software that can be described by certain characteristics.

At this stage he definition of the development model is preliminary and serves as a guideline. The following proposed definition is a minimal definition that only defines obvious properties that makes this way of developing software distinct. It is proposed that a software development project will be defined as adhering to the GNU/Linux development model if the following are true:

The product of development is software (a computer program).

The software is licensed in such a way that the source code is available, everyone is permitted to use and develop the software further.

Everyone is allowed to contribute to the development.

1.1.2.2 Aim of the development model

The GNU/Linux development model must describe how software is developed in the GNU/Linux community. The development model must help the understanding of the development process and details regarding the development process itself.

In the following distinction between three elements are presented. The distinction is introduced to ease understanding of the developing process by dividing it into three separate processes. These processes can then be examined and described separate and subsequent be analysed as a whole.

The three processes:

1. How software (the specific program) change over time. The technical process.
2. How the method of developing software evolve over time. The organisational process.
3. What are the incentives for participating. The personal process.

1) Refers to the actual coding process where program code changes. This is where one or more individuals changes code in the same program within a period of time. As a result, features are added, bugs are fixed and changes are made to the code - the program evolves.

2) Refers to the relations between developers that exist in a software development project. In the beginning of a software project it is usually the maintainer that does everything in the project, code, document, test, develop new ideas and other. When the project develops a developer and user base the maintainer begin to receive feedback. The feedback consists of the odd comments, bug reports, suggestions and fixes.

3) Refers to that incentives and motivation of the individual programmer.

The above mentioned distinctions may prove to have bearing on each other, the distinction, however will ease the identification of properties that are should be ascribed to the development model.

1.2 Research questions

In the previous section it has been identified that the development of the GNU/Linux operating system have been and still is carried out in an intriguing manner. The way Linux and the GNU tools are developed along with related projects seems to share common characteristics. Without further analysis it was proposed that these common characteristics define a certain way of developing software - a development model. The development model will carry the following working title: The GNU/Linux development model. This thesis will answer the following questions:

1. ***What characterises the GNU/Linux development model?***

2. ***What are the economic mechanisms of the GNU/Linux development mode? Sub question: How does existing theory of the economics of innovation apply?***
3. ***What are the implications of using the GNU/Linux development model in a software development project?***
4. ***Does the GNU/Linux development model apply to other than software?***

1.2.1 What characterises the GNU/Linux development model?

Using the framework provided in the above description of the aim of the development model the following three processes must be analysed and characterised:

- How software (the specific program) changes over time? The technical process.
- How the method of developing software evolves over time? The organisational process.
- What are the incentives for participating? The personal process.

It must be emphasised that the GNU/Linux development model centres around the development of the GNU/Linux operating system and the development of related software.

This question is motivated by the fact the GNU/Linux development model has been much applauded in the media. Linux and Open Source have become household terms that seem to have wide appeal. A definition and precise understanding of what characterise the GNU/Linux development model have yet to be offered and are much needed to qualify any further discussion.

The GNU/Linux develop model have been discussed within the community and papers have even been written describing the merits of this special way of developing software. These views have been subject of extensive citing in the media when the development model have been presented and in general when the development model have been discussed. Alas there have already been formed an implicit understanding and definition of the model within the community. Subsequently the media and society have adopted these understandings in general. These views are strongly embedded in the community, which is demonstrated by the fact that random people provide the same answers for question using the same arguments.

Maybe the answers they provide are not correct but the result of a widespread belief.

The question must thus be answered in two stages. The first will characterise the development model using sources provided by the community. This will document the self-understanding of the community. The self-understanding will be verified in the community.

The second stage will be an analysis of the self-understanding where the different element will be identified. This stage will result in a precise definition of the GNU/Linux development model.

1.2.2 What are the economic mechanisms of the development model?

This is a comparative study of the economic mechanisms in the GLDM and the mechanisms of a normal economy.

There are a number of transactions taking place what is their nature and what is their economic content. It must be possible to gain an economic understanding by contrasting the "free" transactions with the commercial transactions of the normal world.

1.2.2.1 How does existing theory of the economics of innovation apply?

It seems that the nature of the innovation in the GNU/Linux development model is different from the areas usually associated with the economics of innovation. This question will address this issue.

As presented in the paragraph 1.2 Theoretical motivation the standard body of theory on the economics of innovation seem to have problems when analysing that GNU/Linux development model. This question will be answered by a brief discussion on the usability of conventional theory to understand the development model.

1.2.3 What are the implications of using the GNU/Linux development model

Doing things in a special way often provide results of a certain type, using a hammer might be good for nails but less successful when tightening nut and bolts. The same thing applies to development models where one type of development model is successful in producing software with certain characteristics.

The aim is to characterise the types of development the works well with the GNU/Linux development model

It must be analysed whether software resulting from the GNU/Linux development model share common characteristics that can be attributed to the development process.

This question will be answered using hypothetical deductive methodology to form questions that will verify the proposed hypothesis. Selected theory must be applied to deduct hypothesises.

1.2.4 Does the GNU/Linux development model apply to other than software?

This is a thought question in the sense that the question will not be exposed to a large empirical study. The questions will be answered by a theoretical analysis and discussion weather it is possible to imagine a physical good being developed and distributed the same way as software in the development model.

1.3 Expectations

En form for formulering af theser eller forventninger til hvad der skal komme ud af dete studie. En form for froventede resultater.

Jeg regner med at Raymond er fuld af l…

Det er ikke et demokrati men en mængde små paver.

1.4 Boundaries of the report

What not to examine, draws a line in the sand.

This thesis has no interest in the technical aspects of programming.

1.3 The structure of the report

A description of the structure of the report and how the reader should understand the different chapters.

1.5 References

Cargile, Carl F. (1989),
"Information Technology Standardization: Theory, Process, and Organizations",
Digital Press, Bedford, MA.

Corbató, F.J. & Vyssotsky, V.A. (1965),
"Introdustion and Overview of the Multics System", 1965 Fall Joint Computer Confrence.

Honderich, Ted (ed.) "The Oxford Companion to Philosophy", 1995, Oxford University Press.

Mahoney, Michael S. (1998),
"The UNIX oral history projekt", AT&T Bell Laboratories.

McKusic, Marshall K. (1999),
"Twenty Years of Berkeley Unix - From AT&T Owned til Freely Redistributable" in "Open Sources - Voices from the Open Source Revolution", Ed's Chris Dibona, Sam Ockman and Mark Stone, O'Reilly & Associates 1999.

Lewis, Pierre P. (1994),"A very brief look at Unix history",
http://www2.shore.net/~jblaine/vault/plh.html.

Leonard, Andrew (1998)
"The Richard Stallman Saga, Redux", Salon Magazine, Sept. 11, 1998
http://www.salonmagazine.com/21st/feature/1998/09/11feature.html

Munkholm, Hanne; Troelsen, Caper; Jensen, Eva D. (1997)

"The Linux Way", rapport made for the Operating Systems course at the Copenhagen Engeneering School (Københavns Teknikum).

OpenSource.Org (1999),
”History of the Open Source effort”, http://www.opensource.dk/mirror/history.html.

Perens, Bruce (1999)
“The Open Source Definition” in "Open Sources - Voices from the Open Source Revolution", Ed's Chris Dibona, Sam Ockman and Mark Stone, O'Reilly & Associates 1999.

Ritchie, Dennis M. (1984),
“The Evolution of the Unix Time-sharing system”, AT&T Bell Laboratories Technical Journal 63 no.6 part 2, October, pp. 1577-93

Ritchie, Dennis M, (1996),
“The Development of the C Language”, Bell Labs / Lucent Technologies.

Salus, Peter H. (1994),
“A Quarter Century of UNIX”, Addison-Wesley Publishing, Inc.

Simonsen, Keld Jørn (1999),
Presentation at the Skåne Sjælland Linux User Group, 12/10-99, www.sslug.dk.

Stallman, Richard (1999),
“The GNU Operating System and the Free Software Movement” in "Open Sources - Voices from the Open Source Revolution", Ed's Chris Dibona, Sam Ockman and Mark Stone, O'Reilly & Associates 1999.

Valloppilli, Vinod
“The Halloween document” an internal Microsoft memorandum leaked to Eric S. Raymond who subsequently published an annotated version at http://www.opensource.org/halloween/halloween1.html.

Quarterman, John S. & Wilhelm, Sussane (1993),
“Unix, POSIX and open systems - The open Standards Puzzle”, Addison -Wesley Publishing Company, Inc. 1993