Intellectual Property Rights in Software
A Critical Investigation from an Ethical Perspective

by

Axel Schulz

Supervisor: Professor Göran Collste

Linköping, May 2004
Intellectual Property Rights in Software - A Critical Investigation from an Ethical Perspective

Axel Schulz

Abstract
The development of software was considered until the beginning of the 1990th as a cathedral like product development in closed companies. This way of development changed in the last decade. Open source software (OSS) development challenged this consideration significantly. OSS is produced in co-operation by skilled people, distributed and used by many moral agents. The result, the software itself, can be studied and modified. Herein is the main incentive for people to develop the software. In such a mode of production the freedom to access knowledge and information (=source code) is a necessity to produce the artifact (software). Software is a digital entity. The main difference in comparison to natural resources like oil, land, minerals is that it can be used and reproduced without losses. It lacks the capacity of getting naturally scarce. Contemporary intellectual property rights assume implicitly that goods might getting scarce one day. Imbedded in the term intellectual property is also an idea of “fencing” objects. In this thesis I will argue that an artificial “fencing” of digital objects might cause unintentional bad consequences for the society. An other quality intellectual property rights are claimed to have is that they serve as an incentive for inventors/authors to produce new inventions and ideas. The practice of OSS development works without such an incentive provided by intellectual property rights. The moral conflict, which I attempt to unravel in this work deals with the question to what extend the application of intellectual property rights in software is necessary and how restrictive particular property rights in digital objects should be – if there should be any at all. Knowledge as the factor of production is of the same value in knowledge societies as land was for agrarian societies. The difference is in the mode of production and the un-limitless availability of digitalized knowledge. I argue that the “protection” of knowledge, and software is knowledge, has to be carefully revised in so called knowledge societies.
<table>
<thead>
<tr>
<th>Nyckelord</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyword</td>
</tr>
<tr>
<td>Intellectual Property Rights, Open Source, Software, Tragedy of the Commons, Tragedy of the Anticommons, Internal Values of Practices</td>
</tr>
</tbody>
</table>
Abstract

The development of software was considered until the beginning of the 1990th as a cathedral like product development in closed companies. This way of development changed in the last decade. Open source software (OSS) development challenged this consideration significantly. OSS is produced in co-operation by skilled people, distributed and used by many moral agents. The result, the software itself, can be studied and modified. Herein is the main incentive for people to develop the software. In such a mode of production the freedom to access knowledge and information (=source code) is a necessity to produce the artifact (software).

Software is a digital entity. The main difference in comparison to natural resources like oil, land, minerals is that it can be used and reproduced without losses. It lacks the capacity of getting naturally scarce. Contemporary intellectual property rights assume implicitly that goods might getting scarce one day. Imbedded in the term intellectual property is also an idea of “fencing” objects. In this thesis I will argue that an artificial “fencing” of digital objects might cause unintentional bad consequences for the society. An other quality intellectual property rights are claimed to have is that they serve as an incentive for inventors/authors to produce new inventions and ideas. The practice of OSS development works without such an incentive provided by intellectual property rights.

The moral conflict, which I attempt to unravel in this work deals with the question to what extend the application of intellectual property rights in software is necessary and how restrictive particular property rights in digital objects should be – if there should be any at all. Knowledge as the factor of production is of the same value in knowledge societies as land was for agrarian societies. The difference is in the mode of production and the un-limitless availability of digitalized knowledge. I argue that the “protection” of knowledge, and software is knowledge, has to be carefully revised in so called knowledge societies.

Keywords: Intellectual Property Rights, Open Source, Software, Tragedy of the Commons, Tragedy of the Anticommons, Internal Values of Practices
To my parents
Sylvia & Jürgen Schulz

and

To my girlfriend
Doreen
© 2004 by Axel Schulz

This thesis is published under the Creative Commons Attribution-NonCommercial-ShareAlike 2.0 license in order to ensure the freedom to access and study this document.

The terms and conditions of the license can be accessed at http://creativecommons.org/licenses/by-nc-sa/2.0/legalcode

Acknowledgments

First of all, I would like to thank my supervisor Professor Göran Collste and express my gratitude for his patience and his enormously important and competent constructive criticism throughout the work. The academic year at the Center for Applied Ethics was a real challenge for me and I am deeply thankful to a number of people who made this year so valuable. Ulrica Engdahl, helped me not to forget that I am in Sweden by some cups of coffee. Tack! Martin Andersson helped me shaping my mind about the crucial questions of the topic of my thesis at some “accidental-seminars”. Tack, Martin. I would also like to thank Professor Bo Axel Petersson, who provided me with a profound amount of knowledge in philosophy in the first term of this year.

I am also grateful to my fellow participants of the Master Program. The intercultural education, the exchange of different views and ideas was at least as valuable for my personality as the academic education.

A huge debt of gratitude I owe my Swedish host family, Birgitta and Bernd Mathiesen. Their hospitality and encouragement provided the best medicine against homesickness and turned my stay in Sweden into real pleasure.

I would also like to thank all the people who helped me to remind that there is also another life than studies. I cannot name all of you here. I just want you to know that I am grateful for your intended and unintended support.

I am also very grateful to the participants of the “discussion” maillinglist at FSFEurope.org. Thank you so much for your patience, and your comments on my rather unqualified questions.


Axel Schulz
Linköping, Sweden, May 2004
# Table of Contents

## Acknowledgments

## 1 Introduction

## 2 The Scope of Software

2.1 Open Source Software  
2.2 Effects of Software  
2.3 Standards and Standards  
2.4 Possible Tasks of an Ethical Analysis

## 3 The Nature of Software and Intellectual Property Rights

3.1 The Nature of Software  
3.1.1 Source Code and Binary Code  
3.1.2 Mass Market and Specialized Software  
3.1.3 Proprietary and Open Source Software  
3.2 Intellectual Property Rights  
3.2.1 The Purpose and Justification of Intellectual Property Rights  
3.2.2 Trade Secret Law  
3.2.3 Patent Law  
3.2.4 Copyright Law  
3.2.5 Concluding Remarks on Intellectual Property Rights

## 4 Ethical Aspects of Open Source Software Development

4.1 The Cathedral and the Bazaar  
4.1.1 The “Community”  
4.1.2 The Social Context of Open Source  
4.2 Voices from the Grass-Ro ots  
4.2.1 Freedom and Power  
4.2.2 Copyright / Authorship / Ownership  
4.2.3 Software is Speech  
4.2.4 Final Remarks on the Voices from the Grass-Ro ots

## 5 Consequences of Ownership and The Value of Virtues

5.1 Commons and Anticommons  
5.1.1 Hardin’s “The Tragedy of the Commons”  
5.1.2 The Tragedy of the Anti-Commons  
5.1.3 Conclusions from the Tragedy of the Commons and the Anti-commons  
5.2 The Internal Values of Intellectual Work  
5.3 Software Development and Virtues
# TABLE OF CONTENTS

6 Another View on Intellectual Property Rights in Software 69
  6.1 Summarizing the Arguments ................................. 69
  6.2 Justification of a New Approach to Intellectual Property Rights in Software ........................................ 75

7 Bibliography .................................................. 77
1 Introduction

The technology software became a major concern of applied ethics in the past 10 years. The major concerns were, for example, how to decide under which circumstances it is morally right to copy software, how ownership in software can be morally justified and by which specific means software can and should be protected in order to protect the original author from exploitation.

Software is considered as a result of intellectual labor. Therefore, the legal means for software protection were derived from other, already existing, forms of intellectual labor protection. Since the early 1980 copyright law, patent law and trade secret law were applied to software in order to enforce the exclusive rights of the authors or inventors.

This legal practice established itself by this time and remained unquestioned until the beginning of the 90th of the last century. What happened? The bundling of communication devices, the improved telecommunication infrastructure and the affordability of personal computers for private persons were the reasons for the success of something we call today the internet. First, the internet was used at universities around the world. The distribution of scientific results and cooperation were technically getting easier and faster. Soon, the new technical entity became a mass medium.

This development also affected the value of software and its protection. "Software was recognized as something with enormous market value, and hence, all the ethical issues having to do with property arose." (Johnson, 2001, p. ix)

By the time the internet became a mass medium, cooperations on the internet increased enormously through the means that the medium provided. The cooperation model which is in the center of this thesis is open source\(^1\) software development. Software were developed in the "internetless" world in private companies by computer engineers who developed the software for the profit of the company. Producing software is/was a business. Open source software development as such is not. It has an enormous monetary effect but the development, like in private companies, is different and aims not necessarily at profit.

\(^1\)Please, allow me to use this term "open source" without any further elaboration, for the moment. Later, I will clarify how we have to understand "open source" software development.
This thesis will investigate how this new way of software development requires a re-evaluation of intellectual property rights applied to software in a “wired world” (Lessig, 2001). The thesis will try to reveal the hidden links between the technology software and the moral justification of intellectual property rights.

The Problem

In a “wired” world the status of information and knowledge changes. The tendency of the “de-embedding”, as described by the British sociologist Anthony Giddens (1991) in his Consequences of Modernity, of social relations and the analogous growing importance of the access to knowledge and information are questioning the application of restrictive and exclusive property rights in non-exclusive objects like software. One of the major problem an ethicist has to address is, if a society is better off if authors enjoy restrictive rights of their creations, or if the society is better off if the actual usability of these creations is possible without rigorous restrictions.

The serious problem, which also this thesis problem will not solve, is how the usefulness of creations and inventions should be measured. While not solving this question this thesis will suggest some possible solutions and point to lines of arguments which have been underestimated so far.

Kai Kimppa (2004a), for example, presented in a recently published article a new approach to intellectual property rights in “digital distributable media”. So far, the determination of maximizing the good or “happiness” for society through intellectual property rights have been measured by quantitative, easy measurable, indicators (e.g., money, amount of registered patents). Kimppa argued, that the qualitative indicators had been underestimated.

I hold, that the greatest amount of qualitative innovations and improvements is the only standard to measure if exclusive property rights in intangible things should be applied or not. The assumption that innovations are possible without restrictive rights of exclusion is founded and justified by the success of the open source movement. It should be noted, that this assumption is opposed to a commonly held view that the only incentive for innovation are rights for the author to exclude people from the usage of is creations/inventions.

The dialectical interdependency between the contemporary status of intellectual property rights together with its widely accepted moral justification and, on the other hand, the line of arguments for no property rights in software, causes the major difficulties for a proper formulations of terms throughout the thesis.

The Material and the Method
The material for this thesis consist mainly of scientific journal articles and other recent writings about the concept of intellectual property rights and its application to software.

It should be noted that I tried to consider the most recently published material from different fields (law, applied ethics, and writings by software specialists).

The method can be described as an analytical investigation of different arguments by different disciplines about the topic of this thesis. Deborah Johnsons (2001) in her coursebook *Computer Ethics* as well as Richard Spinello (1995) mentioned explicitly that ethics and law are heavily intertwined in the debate about intellectual property rights. It should be self evident that this thesis will not provide a solution to this problem. However, I tried to be as specific as possible in the application of the terms.

**The Disposition of the Thesis**

In the second chapter of this thesis I will introduce and illustrate which scope software has and will have in the future in society. This is essential in order to understand why software must have a particular status as a technology in society.

The third chapter will clarify what software actually is, or how software can be categorized (technical, economical). In the second part of this chapter I will present the basic ideas for the justification of intellectual property rights in software before I turn to some specific legal concepts for the protection of software.

Chapter four is concerned with the model of open source software production. Here I chose to present the approach of skilled software engineers to work on open source software. I hold that this presentation will offer the reader, who is not familiar with this way of software development, a good overview and a “pool of arguments”. I submitted additionally a discussion from a maillinglist about the “ownership in software” by open source software developers.

The fifth chapter will provide some arguments about the effects of the application and non-application of property rights to different goods. This chapter will also contain an illustration of the external and internal values of intellectual work.

The last chapter of this thesis summarizes and analyses the main lines of arguments form the previous chapters. Here I will also try to formulate a possible conclusion of my research.
Software will become a lot more important in the future than it is already today. The evidence for such an argument can be derived from the computerization of everyday-life. TV and radio broadcasting is already possible through the internet. Other devices, so far mobile or non-mobile, will make use of the internet in the future. Information on the net will be accessible from almost everywhere, irrespective of time and place. Governments will soon or later require its citizens to negotiate their issues with their official agencies over the internet. Much more people will be employed in the computer industry and related service agencies as there are nowadays.

Of course, where there is so much light there are also a lot of shadows. It is obvious that the privacy and the integrity of people will be more threatened than today. Security issues will play a big role for the administration of huge database networks. Last but not least the accessibility of the internet for every citizen will be a fundamental basic requirement in order to take part in society and public debates.

But these are the obvious and large scale consequences of many technologies which are intertwined and work together as the so called “internet”. What is at stake in this thesis is not the internet as such but one of the technologies which makes it possible: software.

Software has many branches. To talk about software as such would truly underestimate the characteristics of it.

A new kind of software, as I may be allowed to call it, emerged in the beginning of the 90th of the last century at the software development horizon. Its origin can be tracked back to the 1970th. But it was less influential in all the years until the beginning of the 90th. The internet with all the accompanied technologies opened up a new possibility for computer engineers: open software development on the net, practically non-stop and over long distances. The new kind of software is named in relation to the requirement of the engineering process: open source.

This way of “engineering” became very influential in the beginning of the new mil-
lennium because of its economical effects and the embedded spirit. Some developers are involved in these projects because of purely altruistic reasons. They want to push a project which they hold worth while to support. Others support it because they gain an economical value from it – they offer services to customers due to this software. On the next pages I would like to say some more words about open source software and its many implications. Than I would like to broaden the view on other technological and social “effects” of software which emerged in the last years and which are worth while to consider here. In the end of this chapter I would like to summarize the most important aspects and expose the hidden links between them.

As I mentioned in my introduction, software should be considered as much more in value and scope than just as a product. Open source software has a moral impact – not only because of its way of engineering. This impact is relevant for reflections about the concept of intellectual property rights and private property in general.

### 2.1 Open Source Software

With „Why You Should Care“ starts the preface of the influential book *The Cathedral and the Bazaar* by Eric S. Raymond (2001, p. xi) – a leading advocate of the world wide open source movement. The Cathedral and the Bazaar enables the reader to understand why open source software became so influential in software engineering in the last decade and more than this: why it works (what is probably more important.).

The point of open source is not to make the world a better place, as sometimes mentioned by opponents of the movement. This is a more or less unintended side effect. The goal is to provide more reliable, secure, and efficient software. “Open source promotes software reliability and quality by supporting independent peer review and rapid evolution of source code.”

“The idea of open source has been pursued, realized, and cherished those thirty years by a vigorous tribe of partisans native to the internet. These are the people who proudly call themselves “hackers” – not as the term is now abused by journalists to mean a computer criminal, but in its true and original sense of an enthusiast, an artist, a tinkerer, a problem solver, an expert.” (Raymond, 2001, p. 7f)

Before I turn my attention to the definition of software I would like to define what the essence of open source software is. Some authors mentioned that one can use the term “open source” and “free” software interchangeable. This is incorrect (Gacek et al, 2004) and at the same time highly debated within the community. Free software has the characteristic to remain free no matter what. Crucial for “protecting”

---


3 “Free” and “freedom” refers to the freedom to (1) use, (2) study, (3) improve, and (4) redist
intellectual work in software is an appropriate license. In fact, there had been a huge amount of open source licenses popped up during the last years. But only one license ensures the four freedoms, intended and mentioned by the Free Software Foundation. This license is called GNU General Public License. It ensures that a software has to be distributed with the source code and that there should be no restrictions for distributing the software. This last part of the license (no restrictions for distribution) causes the main questions and confusions regarding free software. No restriction means, of course, that one may also sell a GPL’ed software. Thereby, the distributor may only charge a fee for distribution costs but not for the software itself. This restriction is ensured by the license. So the distributor decides about the amount of the fee independently from the author of the software. On the other hand, the “customer” who pays the distribution fees can make as many copies of the software as she likes and re-distribute them free of charge for other users. The GPL explicitly allows such procedure. Indirectly, this clause in the license ensures the community from exploitation.

Richard Stallman is the author of the GNU General Public License and the founder of the Free Software Foundation which enforces the license and tracks violations of software published under the GPL. His main goal is

“... spreading freedom and cooperation. I want to encourage free software to spread, replacing proprietary software that forbids cooperation, and thus make our society better. [...] I figure that since proprietary software developers use copyright to stop us from sharing, we cooperators can use copyright to give other cooperators an advantage of their own: they can use our code.” (Stallman, 2003, http://www.gnu.org/philosophy/pragmatic.html; accessed 03/22/2004)

But Stallman also noted that “If you want to accomplish something in the world, idealism is not enough—you need to choose a method that works to achieve the goal. In other words, you need to be ‘pragmatic.’” (cf. Stallman, 2003) And he was. The purpose of the GPL was to ensure that software, once published under the GPL, remains free. Stallman refers to this as free in free speech, not as in free beer. Restricting someone to sell free software would be an unacceptable burden for distributors etc. That is why the GPL allows selling.

This issue causes of course a lot of debates in business and the community itself because of the small word “free” in free software. The economic effect is of course easy to foresee. The price of distribution cost will remain very low since there is an unlimited supply of the product and since every person can become a “distributor” without asking for permission. After all, the market will take care of the price.

\(^{4}\) GPL’ed — this shortcut is frequently used to express that the software is published under the General Public License.
Derived from these facts one can say that a free of charge free software is a side effect of the license. Because of the misinterpretation of “free”, free is often mistakenly understood and mixed up with free of charge. However, this is an important distinction one has to keep in mind when one talks about the consequences of free software for society in its ethical impact on it.

Now I am going to describe why there is a difference between open source and free software. The term “open source” can be understood in two ways. On the one hand, open source has a technical meaning in the intelligence community; it refers to publicly accessible intelligence sources such as newspapers (cf. http://opensource.org). On the other hand, open source may refer to the availability of the source code. This is the intended expression of open source. Meanwhile an initiative emerged at the horizon of software engineering which trademarked the label “open source”: the Open Source Initiative (OSI). The goal of the corporation is to promote open source software. Thereby that do not pay much attention on particular licenses used for such software. The only requirement a software license has to match is to declare the software under it as open source. That means the source code of the software has to be accessible.

Some licenses may contain "copyleft" restrictions requiring that the source code must always be made available, and that derived products must be released under the exact same license.

“The GNU GPL is not Mr. Nice Guy. It says “no” to some of the things that people sometimes want to do. There are users who say that this is a bad thing—that the GPL “excludes” some proprietary software developers who “need to be brought into the free software community.” But we are not excluding them from our community; they are choosing not to enter.” (Stallman, 2003). What Stallman stresses here is the “viral” effect of the GPL. Software, which makes uses of already GPL’ed software has to be published under the GPL.

About what kind of software are we talking? I think that Gacek et al. (2004, p. 7) made a very good point when she said that developers are always users. This implies that open source communities are developing code which is somehow demanded — either by users or by developers. Linux was, for example, a demand by developers. Its purpose was to develop a operating system which is Unix based and freely available. The OpenOffice.org-project (OOo), on the other hand, was a user demanded project. The purpose of OOo is to serve the needs of end-users in everyday life.

In contemporary philosophical debates about software (Spinello et al., 2003a,b) it is very obvious that software is considered as a tool which serves a need or a de-
mand. Besides other things, the fact to write an academic paper may demand the
development of word-processors, like Open Office, or the more well known Microsoft
Word. The results of the software development processes are considered as products,
expressed in pure commercial terms.

I think one can refer to these kinds of software as the typical Graphical User
Interface (GUI) applications. This is what the non-experienced end-user is looking
and asking for. However, besides this kind of software I would like to distinguish
non-GUI applications. This kind of software rules and keeps the internet alive. E.g.,
is the email software which directs our every day email through the net a typical non-
GUI software. Mostly, the end user never gets in touch which this kind of software.
She does not have to care about its installation and configuration. It simply works
and is somehow there. The end-user is mostly unaware that she demands or needs
this particular kind of software. The Apache web server, for example, runs nearly
70% of all websites on the internet\footnote{I use here the official and representative Netcraft survey for web server.
This survey is updated regularly and can be accessed through the URL http://news.netcraft.com/archives/web_server_survey.html. I accessed this site on 02/19/2004.}. The purpose of this software is to serve from a
client (end-user) requested files. That means, if you type the address of a website,
Apache will take care of your request and distribute the requested page (files) to
you. Users are mostly unaware of these kind of software which serves their needs so
smoothly and free of charge. However, this software makes the internet work as it is.

One also has to note, that different software programs can fulfill one and the same
task. The task of the Apache web server is to serve pre-defined folders and files on
request. This task can also be fulfilled by an Rexen web server or a Microsoft web
server. There is. Indeed, a huge pile of software which can perform one and the
same task. What matters (or at least should matter) in the end to the end-user
of the software is that this software works properly. So it is important to keep in
mind that tasks and software can be performed by different applications. This will
become an even more powerful argument, when I will later on refer to standards on
the internet or in software engineering in general.

Software enables the flow of information and enables people to do things which were
impossible before. In fact, it is the software which opens up certain possibilities. Let
us take two examples. A word processor like Open Office enables the user to create
files. These files are still electronic and allow the user to do whatever she likes to
do with them. One can use the software to write a Master thesis, like I did in this
case. But you may also use it to create Happy Birthday cards or whatever you like.
So, software is capable of producing software. Because as long as one does not print
the produced files it remains literally spoken soft-ware. But is this software (output file) of the same quality than the application by which it was created?

The second example I would like to mention here is file-sharing software. It enables the user to share information in a digitalized form over the internet. Music, software, writings, even whole movies can be shared\textsuperscript{6}. Now, is there a difference between these two kinds of software? And in what way do they differ from the web server software? As we can see there is a huge variety of software and purposes it is used for or can be used for. One can also easily see that they have quite important tasks. At least, I consider my MA thesis as quite important. But it is also important to ensure the accessibility of websites by every person who requests them. Accessing websites means to access information and knowledge. This will be in the future even more true than it is today. Eric S. Raymond (2001, p. xii) wrote that this indicates the "information-rich post-scarcity economies of the 21\textsuperscript{st} century”.

\section*{2.2 Effects of Software}

A very important point one also has to keep in mind is the diminishing necessity of distributors. The internet allows people to download music from the web. And I do not mean the illegal file sharing activities but internet portals like Apple’s iTunes, for instance. But not only in the "old-fashion” media world the importance of distributors gets less and less important. The same holds for the software world. Huge companies like Macromedia Inc., Microsoft Inc., SUN, allow the download of software directly from the producers websites. The world-famous success story of the free operating system Linux would be impossible without the means of the internet. At kernel.org everyone can download the core of the Linux operating system. It is simply there and with ease to access: 24 hours a day, 7 days a week. And it does not diminish or lose quality by the previous download by other users. Additionally I would like to mention here that other distributing systems emerged in the last years. The open source world set up certain websites which host databases which contain information about various open source software. Practically everyone can access these sites and download the appropriate software which serves one’s needs.

\textsuperscript{6}I am not going to discuss here the legal aspects of file sharing. This would be beyond the scope of this thesis. I just want to refer to recent court decisions in Norway regarding a hacking an encryption system which protected a DVD from copying. The judge decided that providing a software which is capable of hacking a copy-protected DVD is not against the law. While using this software for hacking those CDs and distributing the hacked files on the internet is. (cf. http://cryptome.org/wipo-imp99-3.htm) A proper discussion on such issues would really require an other thesis.
One of the most famous and well known sites is SOURCEFORGE.NET (or SF.NET). The financing is ensured by sponsoring for this site and advertisement on the site. Meanwhile, new premium services on SF.NET popped up. Now, one can get access to very advanced search capacities in order to get much faster a much better result about the software she is looking for. Still, one has to keep in mind that this service is just a service and not a necessary condition in order to get the software. Sourceforge has no rights in the hosted software. It is more a search engine for free available open source software. This is the major and crucial difference in comparison to other “old-fashion” distributors (like hard-ware publishers of books, CDs etc.).

Georg Greve, president of the Free Software Foundation Europe, makes that point very clear when he said that we do not need any more such “old-fashion” distributors.

But why are all these empirical appearances/evidences worth mentioning? Where is the ethical question in all this? Why should someone care? As I have mentioned earlier software gets very important. Society and human relations will be shaped more and more by this means. Individuals will have to rely on this new tool. In order to gain membership in pure online communities on has to have access to these internet.

As I said in the introduction, by using contemporary articles dealing with the question of property rights in software I would like to investigate if the concept of intellectual property rights is applicable to software. This contains the question if software should have owners at all. I hold that such an investigation will be very fruitful for the ongoing discussion on intellectual property rights in software in the future. After studying and reviewing some of the relevant literature in this field I found out that the voice from the open source movement are underestimated in contemporary writings in the field. This is a real pity since they offer a completely new view on intellectual property rights. The most often mentioned reason why intellectual property rights are useful is to ensure progress in society. I hope to be able to illustrate that this view is mistaken when it comes to software. The task applied ethics can fulfill here is to offer reasonable arguments for such a claim. I hold that software is a means to social goods\footnote{I do not will go in detail here into a theory of “social goods” but “knowledge”, for example, was considered as a “social good” by Martin Walzer. That is why I use the term here.} (knowledge, membership). Any form of restriction of software development which prevents free open source development will decrease the use of software for the commons and the access to information.

I order to find a common ground from which I can build my arguments from I will first of all go on and investigate the concept of property rights as such and the relevant branches of intellectual property. I hold that some conclusions which
had been accepted for other kind of properties as valid are weak and inapplicable to software.

I think it is unavoidable to choose an utilitarian way of arguing and weighing the the main positions against each other. I think it is obvious, that many different moral values are embedded in this issue. One of the main questions will also be who decides about ownership in software? Should a society allow ownership in software or should it be up to the engineer to decide about it? What if one society grants intellectual property rights in software but another doesn’t? Since software is worldwide available on the internet it is hard to track violations of property rights.

Another question is the impact software has on other technological developments. For instance, is Trusted Computing a highly debated in Europe and the United states at the moment. Trusted Computing (TC) is intended to make the computer “save”. Save in the sense of disobeying you:

“Trusted Computing […] the plan is designed to make sure your computer will systematically disobey you. In fact, it is designed to stop your computer from functioning as a general-purpose computer. […] The technical idea underlying treacherous computing is that the computer includes a digital encryption and signature device, and the keys are kept secret from you. Proprietary programs will use this device to control which other programs you can run, which document and data you can access, and what programs you can pass them to.”

(Stallman, [1])

Digital Rights Management (DRM) is another highly debated issue. DRM is supposed to ensure and to enforce the rights of authors.

“Hollywood and the record companies plan to use treacherous computing for DRM, so that downloaded videos and music can be played only on one specified computer. Sharing will be entirely impossible, at least using the authorized files that you would get from those companies.”

The rights of authors of any kind of art – software included. From 2006 onwards hardware in the United States has to be distributed with a working DRM system. This system is capable of denying the owner of the PC to make copies of DVDs, CD ROMs, Software of any kind and so on. The system is realized by fixed (un removable) software on hardware. Every home personal computer has to contain such hardware.(cf. Stallman, [1]) What is important in this paper is that free software – as mentioned above – wont run on this hardware. This is the hidden link between all the digital rights debates and arguments by the open source software engineers.

Now, let us turn back to the DRM problem: But what if the author does not want this limitations for his original kind of authorship, or, the DVD is not usable on Linux operating systems? In a recently finished trial the Norwegian Jon Lech Johansen

---

8[1] Http://www.gnu.org/philosophy/can-you-trust.html. The term “Treacherous Computing” was introduced by the critics of trusted computing. I will use both terms here interchangebly.

9Another term similar to Digital Rights Management is Digital Restrictions Management.
was declared not guilty because of a distribution of a so-called DeCSS program. This software is capable to crack a copy protection on DVDs. Johansen used this software in order to watch a movie on his Linux operating system. He bought the DVD but he was not able to watch it because of a built-in DRM protection system on the DVD. So, he started programming a small tool (DeCSS) which will knock-out this “protection” system. Afterwards he published this software on his website for other Linux users who had the same problem with watching DRM protected DVDs on their Linux operating systems. Additionally he posted a note on mailinglists, newsgroups, and bulletin boards on the internet.

The court justified its decision because it could not see any violations of intellectual property rights since Johansen only tried to make use of his legally bought DVD. The only purpose of this software is to help other people, as the jury argued. Furthermore, Johansen cannot be held responsible for a misuse of the software, e.g. using it for cracking the copy protection and reselling the copied DVDs.

As we can see, this example illustrates a major clash of interests between people creating and distributing things and people who do not want any (commercial) credit for their creations. But it also illustrates the clash of common and private interests, derived from common and private property. This clash is morally relevant because it the legal practice cannot serve as a moral justification of (intellectual property) rights in software. The legal system of enforcement of property rights is based on pre-internet circumstances. In the German jurisprudence (cf. Lutterbeck, 2003), for example, exists a sharp distinction between the original author of art (Urheber) and the distributor (publisher) of the same piece of art. In pre-internet circumstances this division was useful because the author had no chance of efficient distributing her creation but using a publisher. Of course, this publisher had to have rights in order to risk a distribution of creations by the original author. The worst case in such a system, e.g., was a no-sale of printed books, tapes, LPs, CDs – in short of already-made-material-goods (physical media). In the internet age, such a publisher is not a necessity any more. Even though in the case of books it is, of course, still useful (print-on-demand services disregarded). But when it comes to software, music, or movies the role of a publisher is highly questionable. The author can publish her creation by herself because the new medium internet opens up this possibility. On the other hand there is no risk any more of material losses. The (digital) medium has not be to reproduced – a book has (physical medium). The cost for using the


Pre-internet: I refer here to the circumstances of publishing and distribution before the advent of the internet.
medium and accessing the piece of art has to be paid by the customer. Grassmuck (2003, p. 31) referred to the medium as to the container and to piece of art as the content.

2.3 Standards and Standards

Now, I want to say something about standards in the software world. Basically there are two different kinds of meanings of standard. A white sheet of paper which measures 21,00 mm in the width and 29,70 mm in the length matches the requirements according to ISO norms and is therefore a standardized A4 sheet of paper. Standard understood this way can be defined as something that has to match predefined, public known, and freely available requirements – like the ISO norms.

The Hypertext Markup Language (short: HTML) is the standardized language of the internet. Programers have to learn this standardized language in order to achieve a good result in producing websites. On the other hand do the browsers (e.g. Microsoft Internet Explorer, Netscape Navigator, Opera, Mozilla) have, through which the end-user access these websites be conform to the standards, too, in order to interpret the code of HTML files correctly. Otherwise a website will look totally different in different browsers. Unfortunately, that happens in practice quite often. The producers of internet browsers do not stick to the technical official standard. Instead they try to establish a quasi-standard with commands only a particular browser can read and understand.

But there is also another kind of standard. In huge cities the standard transport vehicle is the subway – the subway as a standard. Standard in these terms means something totally different as in the first example. This kind of standard is best understood, I think, as a standard in the sense of “most often used” or as “best known”. This example can also be applied to the software world. Microsoft Word documents (.doc) also claim to be the standard in the digital office world. Indeed, they are, but more in analogy to the subway example than to the HTML example. Please note that an artifact can of course fulfill both definitions of standard: technical official

---

12 All translations in this thesis from German to English provided without any exceptions by A.S.
standardized and most often used. HTML is such an artifact\textsuperscript{15}.

Now, one may ask why we should have standards at all? Let us find an example (analogy) about standards in the real world. For instance traffic rules. Traffic lights, for instance, are standardized: red, yellow, green. The red light urges the driver of a vehicle to stop at a traffic light. On the other hand this standard is enforced by a legal system and therefore one can talk about a legal obligation to stop at a red light. But the traffic light as such is a standardized tool. It works in many countries in the world. Individuals, traveling through these countries do not have to learn new rules when they enter a new country in respect to traffic lights. This standardization makes sense.

The same is true for the software world. Standards guarantee the burden free communication of devices such as mobile phones or personal computers. Applications can make use of these standards. The most often used standardized protocol of the internet is TCP/IP\textsuperscript{16}. This protocol ensures the communicability from computer to computer. This standard is open an free. Every software engineer can access and study the characteristics of the protocol and use it for her own applications. As we can see, a standardization makes sense here as well.

### 2.4 Possible Tasks of an Ethical Analysis

But do have free software, open standards, trusted computing, DRM anything in common? Why should they be an issue for an ethical analysis? To illustrate that ethics, as a discipline, has plenty of good arguments to offer to this and many other questions will I try to present now.

Some things on earth are scarce or rare. Love, generosity, real friendship, and peace – especially in these days. But these things I do not have in mind. I am more concerned with the physical things which are rare. They are scarce because of natural reasons (oil, water, natural resources in general) or scarce because of artificial limitations (property in something). In respect to the first, human being can do only a few things about it. On the latter one can do much more about it because the rules which makes these things scarce are socially agreed upon. People may agree upon what can be owned or what has to remain in the commons. I am not so much concerned with the different varieties of agreement in different cultures or different countries. These are legal questions which reflect a cultural background. The thesis attempts not to take up questions of moral relativism because this would extend the scope to an unacceptable degree. I am concerned about the moral justifications

\textsuperscript{15}I admit that artifact is a very blurry term here.

\textsuperscript{16}http://www.faqs.org/rfcs/rfc1155.html, accessed 03/17/2004
of ownership or non-ownership in non-natural things, more precisely software. This thesis will explicitly investigate the moral justification of property rights (ownership) in software. The term “intellectual property right” will be questioned. Some claimed that this term is misleading in respect to software because it underestimates the real virtuous/characteristics of software.

What is added in this investigation compared to previous ones is the analysis of property rights in software where software is considered as a very important means to social goods. I hope that this will widen the scope of the contemporary moral justification of property rights in software. I hold that it is not enough to analyze the artifact software separately from its importance in an advanced technological society.

I admit that the task of such an analysis is very ambiguous because it is highly value loaded field. Aristotle said that doing ethics is a “practical’ endeavor which gives us practical knowledge.” (Spinello, 1995, p. 14) But what does that mean? “He meant that one does ethics properly, adequately, reasonably, if and only if one is questioning and reflecting in order to be able to act – i.e. in order to conduct one’s life rightly, reasonably, in the fullest sense well.” (Finnis, 1983, p.1, In: Spinello, 1995, p. 14)

Normative ethics is the established term for these kind of inquiry. Normative ethical inquiry, then, seeks the truth not only for its own sake but also as a basis for choosing proper actions and the right way of life.” (Ibid., p. 15) I do not know if I can fully agree with Spinello. The main problem in doing normative ethics, as I see it, is to achieve a coherent and comprehensive and broadly accepted analysis (and solution?) of empirical problems which contain an ethical question. The ethical question emerges in the empiricism because something goes wrong.

In the case of open source software it is the tread of the generalized concept called “intellectual property rights”. Intellectual property rights like copyrights and patents had been established to protect the artist or inventor from exploitation or financial losses. Patents had been especially successful because they ensured a financial compensation for the inventor if her discoveries were used in industrial productions. In opposite to the original idea of patents the granting of software patents does not protect the free lance software developer but the huge software companies. I will argue that the granting of patents is not any longer an incentive for innovation but a business method. When it comes to software, the rights in intellectual ideas are protected and traded on the market and not the industrial de-

---

17 And only these to concepts will matter in this analysis.
18 Please consult the statistics of the European Patent Office mirrored at http://swpat.ffii.org/patents/stats/index.en.html#jarappl
veloped products (software, applications). The incentive of innovation is secondary. Patents allow the owner to prohibit other competitors to enter the market. The competition and therefore innovation gets stuck. Monopolies of patent-owners will more or less in the future dictate what kind of innovative products will be available on the market. TC and DRM are heading in the same direction. The Johansen-case shows impressively what the claims of the publisher-industry are. The end-user will have to accept an artificial prohibition of exercising his will built-in in the software and hardware he is going to by. The same might hold for standards. Standards in the software industry advanced the development of applications and so on. If standards\textsuperscript{19} will be considered in the future as a threat to intellectual property because a open source application might be able to communicate with an proprietary software then we loose a major useful tool in computer engineering.

The official logic and justification of TC and DRM is to prevent the user from doing something stupid. But should a law prohibit the user from committing a crime when it is highly questionable if committing this crime is morally highly questionable? Shouldn’t one instead revise the legal system? Can cracking a DVD for private purposes be considered as a crime at all? The Norwegian court denied this question!

A very few people are aware of the development in the software world these days but the consequences will effect all of them.

I am well aware that an investigation of the concept of intellectual property is very much depended and intertwined with law. Again, Spinello wrote: “In some areas, such as intellectual property, law and morality can become seriously entangled. Indeed the interplay of legal and ethical issues in intellectual property cases makes them extremely complex.” (Ibid., p. 15). I think this was meant by Finnis.

However, I think that the key to bring a little more light into issue of intellectual property rights and its applicability to software is a careful investigation of the concept of intellectual property rights, and property rights in general. Software is a really important tool for the future. If all the ideas of restrictive legal enforcement’s will be set in the legal system it is doubtful if an open source software development will still be possible. The threat is that a legal system might label OSS as illegal because it does not keep up with TC, DRM, and is violating patents.

As I hope to be able to verify: There are ethical questions involved in the issue of open source software development and contemporary intellectual property rights law. This law is mainly based on philosophical foundations. What I would like to investigate is if the applied philosophical ideas still fit to the concept of intellectual

\textsuperscript{19} Standard understood in analogy to the HTML example given earlier.
property rights applied to software.

As I said earlier, software has to be considered as a means to a social good. If we accept this assumption this may also change the social meaning of software in a society; because social meanings “are historical in character.” (Walzer, 1983, p. 9)
3 The Nature of Software and Intellectual Property Rights

This chapter aims at providing an overview about the definitions and categorizations of software. Software is a very divergent “thing”, which has many aspects. In the literature this diversity has been captured under the term “nature of software” (Davis et. al., 1996, 23) Davis et al. pinpointed to a dual nature of software: it is functional and literary. I will add some more distinctions to this crucial one. Intellectual property rights are applied to immaterial “things”. It is therefore important to determinate what this “thing” is. Software covers, as we will see, a broad spectrum of technical and economical characteristics. Intellectual property rights are applied because of these characteristics. The justification is depended on quality and value of the characteristics. Software, in this case, can be considered as literary work, executable, digital, non-exclusive, open, closed, proprietary, a mass product, a specialized product etc. All these characteristics should matter and should be investigate if one tries to present another view on intellectual property rights.

In the second part of this chapter I will present the legal concept of intellectual property rights; that is its scope and its branches.

Later I will argue that the “thing”, the object, determines if and how intellectual property rights can be applied.

3.1 The Nature of Software

In order to clarify what software “as such” is I would like to quote a definition of “programs” provided by Gemignani. Programs are the technical artifact which can be understood by a machine. This is what also can refer to as software.
Programs are responses to problems to be solved. First, the problem in issue must be clearly formulated. Then a solution must be outlined. To be amenable to implementation on a Computer, the solution must be expressible in a precise way as a series of steps to be carried out, each step being itself clearly defined. This is usually set forth as a flowchart, a stylized diagram showing the steps of the algorithm and their relationship to one another. Once a flowchart has been constructed, it is used as a guide for expressing the algorithm in a “language” that the Computer can “understand.” This “coding” of the program is almost certain to employ a “high level” Computer language, such as BASIC or FORTRAN [Nowadays the most used programming languages are C# and JAVA]. When the algorithm is “coded” in a high level Computer language, it is called a source program. A source program may bear a striking resemblance to a set of instructions expressed in *literary form*. The source program is fed into the Computer by means of an input device, such as a terminal or card reader. The source program is “translated” by the Compiler, a part of the operating Systems program, into machine language, a language not at all similar to ordinary speech. The program expressed in machine language is called an object [binary] program. It is the object program which actuates the setting of switches which enables the Computer to perform the underlying algorithm and solve the problem.” (Gemignani, 1980)

In an ethical paper it would be completely insufficient to accept a technical definition only. To make some meaningful analytical statements about a technology, one has to investigate the impact and the effects for society and, properly most important, how the technology is developed. This shapes the technology. Hence, a technology is not neutral. (cf. Collste, 2001, p. 389)

Software or computer programs are not neutral technical artifacts. They influence the actions we are allowed to take through them etc. This had been outlined in the second chapter. So, let us take a more closer look on nature of the artifact which is at stake here.

### 3.1.1 Source Code and Binary Code

When we talk about software we mostly have the ordinary applications like Windows, Word, Excel[^20] etc. in mind. But this is only the ‘edge’ of software; the part which is visible and somehow useful for the user. It is a unwritten law that every frequently used software is useful. A lot of applications are somehow “hidden” because they lack the capacity to have a graphical user interface. “A graphical user interface (or GUI, pronounced "gooey") is a method of interacting with a computer through a metaphor of direct manipulation of graphical images and widgets in addition to text.”[^21] I want to clarify in this sub-chapter the pure technical distinction of software between source code and binary code:

[^20]: These are product-names and trademarked by the Microsoft Corporation, Redmond, U.S.A.
“Source code and object [binary] code refer to the ‘before’ and ‘after’ versions of a computer program. The source code consists of the programming Statements that are created by a programmer with a text editor or a visual programming tool and then saved in a file. For example, a programmer using the C language types in a desired sequence of C language Statements using a text editor and then saves them as a named file. This file is said to contain the source code. It is now ready to be compiled with a C Compiler and the resulting Output, the compiled file, is often referred to as object code. The object code file contains a sequence of instructions that the processor can understand but that is difficult for a human to read or modify. For this reason and because even debugged programs often need some later enhancement, the source code is the most permanent form of the program.” (see http://whatis.com, search term: software, accessed 04/09/2004)

Another form of a technical distinction can be made between system and application software\(^\text{22}\). Software for personal computers is often broken down into two major categories: system software that provides the basic non-task-specific functions of the computer, and application software used to accomplish specific user-oriented tasks.

System software is responsible for controlling, integrating, and managing the individual hardware components of a computer system so that other software and the users of the system see it as a functional unit without having to be concerned with the low-level details such as transferring data from memory to disk, or rendering text onto a display. Generally, system software consists of an operating system and some fundamental utilities such as disk formatters, file managers, display managers, text editors, user authentication (login) and management tools, and networking and device control software.

Application software, on the other hand, is used to accomplish specific tasks other than just running the computer system. Application software may consist of a single program, such as an image viewer; a small collection of programs (often called a software package) that work closely together to accomplish a task, such as a spreadsheet or text processing system; a larger collection (often called a software suite) of related but independent programs and packages that have a common user interface or shared data format, such as Microsoft Office, which consists of closely integrated word processor, spreadsheet, database management system, etc.; or a software system, such as a database management system, which is a collection of fundamental programs that may provide some service to a variety of other independent applications.

\(^{22}\)From here onwards I will heavily lean on the definitions of software provided by the free encyclopedia WIKIPEDIA at http://en.wikipedia.org/wiki/Software, accessed 05/13/2004.
3.1.2 Mass Market and Specialized Software

The previous distinctions reflects basically the technical side of software. What also matters, and probably is the economic character of the medal, are the market effects of software. Davis et al. offer here a reasonable description of how a economic character of software can be understood:

"First in talking about the software marketplace, we are focusing primarily on mass market software, which basically means personal computing software (though we believe much of what we offer carries over to other software markets). Where product volume is small enough (e.g., mainframe software) to permit it, individual sales contracts can be negotiated to meet the individual needs of buyer and seller. When products are sold in the tens of thousands to millions of units, only standard, agreed-on rules are feasible, rules of the sort provided by intellectual property law."

(Davis et al., 1996, p. 21)

This is a quite important distinction. Mass market software is produced on large scales and in anticipation of the demand or need of the potential customers. The contemporary applications software industry is producing and developing their software in this way.

This has of course an impact on the protection-by-exclusion-mechanism of software. Again, Davis et al. asks: ‘does that mean that mass market software producers have a lobby in enforcing their claims in the legal system while ‘specialized software’ producers, which is mainly open source software, has not that power?’

In short: Our industry lacks an agreed-on set of rules for competition.

Now, I want to propose another set of possible distinctions which may matter for the justification of the application of “right” to software. This distinction reflects the “meta-technical” characteristics of software, as I would name it. Here it does not matter if it is a an application, or a system software or an operating system and what so ever. Instead, it reflects the characteristics software has as such. The distinctions can be listed in as followed: Software is:

- digital
- non-exclusive
- intangible
- executable

The distinction between intangible and digital has to be pointed out because an intangible good does not have to be digital. I admit that the the term “good” is not very suitable here. Let us take an example. A story in a book is an idea. The idea is basically how to put things, e.g., the gardener kills the landlady. This idea is intangible and cannot be owned or restricted. The book, now, contains this idea. The book can be bought and, probably, also has to be bought. But the book can
also be digitalized. That is why intangible and digitalized are unlike.

A software, on the other hand, contains all these characteristics. The problem which has to be resolved in the future is the question how to make such a good protectable. I am aware, of course, of the contemporary practice to apply patent law and copyright law to software. But in order to re-evaluate these protection mechanisms I would like to race the question for protectability completely new. This is what Spinello for instance ask for when he said that philosophers have to “re-examine the underpinnings of the moral legitimacy for intellectual property protection.” (Spinello, 2003a, p. 2). I am not sure if intellectual property rights as such has to be evaluated but instead the applicability to particular goods like software.

Why it is necessary to re-think the applicability of intellectual property rights to software illustrates the following quotation by Davis et al.: Davis introduced in respect to current protection mechanism of software (copyrights, patents) the dual nature of software:

“The inseparable dual nature of software; it is inherently both functional [executable] and literary [intangible], both utilitarian and creative. ... To the copyright scholar, a crucial characteristic of software is its creation in the medium of text; the fact that that text happens to do something useful is irrelevant, because useful behavior is explicitly outside the scope of copyright law.” (1996, p 23)

Indeed, software has a utility-characteristic. It can be a tool to for production. People can actually make use of software and produce papers, spreadsheets etc. with it. One cannot do this with any other copyrightable material.

**3.1.3 Proprietary and Open Source Software**

Another possible categorization of software can be made between the two lines of development (open source vs proprietary).

The open source software model is a way for developing/engineering software outside existing established software companies. This might serve as one of the most important incentives to re-think restrictive and exclusive intellectual property rights. Alongside with this new way of software engineering some authors (Lessig, 2001; Benkler, 2003) started to talk about a new mode of production where knowledge is not produced anymore behind closed doors but on a public medium like the internet.

Indeed, one of the incentives of working on open source software is the possibility to study the source code. Benkler and Lessig noted, that this studying and voluntary reproduction of new market values is the network effect of technologies like the in-
ternet etc. An empirical example might be the free encyclopedias WIKIPEDIA.ORG and proprietary. Benkler noted that the free encyclopedia is growing faster and contains more entries compared to its "lifetime" than any other proprietary because simply everyone person can put or "start" a new entry.

If this is the case, that knowledge and information is voluntary shared without rights of exclusion then applied ethics should turn the focus on this issues and try to offer reasonable answers to the question how a society should re-construct the system of exclusive intellectual property rights.
3.2 Intellectual Property Rights

The term intellectual property rights is a concept in the legal system which describes a set of rights for the protection of ideas, new inventions, information, artistic and literary work—short, every kind of immaterial goods which is of artistic or technical value. I have to leave the question when something is of value aside because this is highly dependent on the moral views of a society, and might vary from one society to another. For my purposes here I will just assume that software contains these values. For instance, the development process of open source software has an internal value, as I would argue. This might matter when a society is going to apply intellectual property rights to software. On the other hand it is not a direct consideration (part) of the legal concept intellectual property in itself, so far. This is one of the reasons why I hold that application of intellectual property rights must be re-evaluated and the full "nature" (e.g., utility for society) of the object (here: software) must be taken stronger into consideration.²⁴

### 3.2.1 The Purpose and Justification of Intellectual Property Rights

In this sub-chapter I would like to introduce, very briefly, the moral underpinnings of intellectual property rights. Afterwards, I will describe three of the commonly used sets of rights for the protection of computer software. I have to say that this part of the chapter will only mention the commonly accepted justifications for private property. A critical investigation of some of the arguments will be presented in the last chapter.

Moral and political philosophers (Aristotle, Locke, Nozick, Hegel) developed reasonable arguments and justifications for the institution of private property in society. All of these authors founded and defined the rights and duties to and for private property on their particular moral/political theory. Now, I want to present some of the arguments which have been put forward for the justification of property rights in objects.

²⁴Titmuss (1970) presented a theory about the donation of blood. His main question therein is if blood should become a commodity or if it should be freely available and the donator should receive no monetary reward in return. Here we can see that a great deal of the moral questions is embedded in the object because itself. A neglecting of blood transfusion because of monetary reason seems to be immoral per se. The object and the principles for distribution seems to have an internal moral dependency.
“Perhaps the most powerful intuition supporting property rights is that people are entitled to the fruits of their labor. What a person produces with her own intelligence, effort, and perseverance ought to belong to her and no one else.” (Hettinger, 1989, p. 36) This is the basic idea in John Locke’s *Two Treatises of Government* about the acquisition of private property. This idea constituted the leading paradigm for the justification of property rights in goods. Every “un-owned” object is capable for property acquisition in this Lockean terms. The idea is sound and have been applied to justify a property claim (right) in objects. An object is acquired when someone mixes her labor with this particular object. The moral justification of such a right is based on the natural entitlement to one’s persons body. A persons owns her/his body and is therefore entitled to use it without external constraints.

Spinello (2003a) claims that if we “accept some version of the Lockean perspective that individuals have a natural entitlement to control the results of their labor it would transfer over to immaterial as well as the material results of that labor.” That is why the Lockean natural rights theory is relevant for the justification of intellectual property rights in immaterial objects. The foundational concept of Locke’s theory is a state of nature. The incentive for people to form a state was expressed in the following way by Peter Singer: “People ... join and remain in society for the advantage they get out of it.” (Singer, 1991, p. 207) This is a quite pessimistic view on mankind because it conceives human being as purely rational. Other theorist have objected to this view that people are born into society and cannot else but form and live in this community. “Rousseau and his successors, on the other hand, see the state more as a community which, in addition to merely providing opportunities for material gains, gives meaning to the individual’s existence and inevitably has a formative influence on the nature of the people who grow up in it.” (Ibid., p. 207)

I think it is easy to see that these two different kinds of theoretical approaches will offer different justification for the setting and distribution of rights in a society. Intellectual property rights are rights of exclusion. A moral/political theory has to justify such a right of exclusion.

What clashes, and what is why an investigation of intellectual property rights is necessary and valuable, is the historical dimensions of rights and the different interest of groups and individuals in a society. Generally speaking it is not only a natural right which might serve as a sufficient justification for property rights in objects. A sufficient claim right might also emerge out of a valid interest claim, a

---

25 Water, for instance, was 20,000 years ago not considered as a scarce good. Today it is, in many parts of the world. This changed, consequentially, the value of water and the moral claim rights in/to water and therefore the value/status of water.
desert claim, or entitlement of a person to something, etc.

"[A]n argument for private property is right based just in case it takes some individual's interest (or the interest of some or all individuals severally) as a sufficient justification for holding others (usually governments) to be under a duty to create, secure, maintain, or respect an institution of private property." (Waldron, 2002, p. 87)

As we can see, a justified claim, a right, depends heavily on the "social arrangements". The difficulty then is to find the most proper way to balance these valid claims. I hold that an utilitarian approach might be the only reasonable way. In the very recently published literature on the topic the utilitarian approach had been favored, e.g., by Kimppa (2004a).

A utilitarian approach provides the capacity to balance claims of different parties to different objects. A more sophisticated discussion on this will be taken up later.

As I said, property rights are rights of exclusion. The valuable task utilitarian theory can provide is to decide to what extent a right should be exclusive. An exclusive right is an right to, for instance, posses, use, manage, waste an object.

Now I would like very briefly present why the social institution of private property is held to be necessity for society. Private property is a bundle of rights, liberties, powers, duties (cf. Waldron, 2002, p. 60). These rights entitled the owner of an object, which he acquired throw his labor, desert, first occupancy etc., to exclude other people from the usage of her legally acquired property (object). The justification of private property can, again, be justified from an utilitarian approach. Aristotle, for instance, “remarks that individual ownership creates a more thorough and stable community of interests, and better promotes efficient, economical, and careful use of things than does common ownership.” (Becker, 1977, p. 62)

In this thesis I attempt, first of all, to investigate the concept of intellectual property in software. But recent writings (Benkler, 2003; Lessig, 2001; Lutterbeck, 2003; Kimppa 2004a,b) may also suggest to re-evaluate the justification of intellectual property rights as such. I am inclined to agree with such an approach. Before I turn back my attention to the specific object of investigation, which is software, I would like to list three forms of intellectual property protection which can be applied to software. This might illustrate how software protection works in the legal system nowadays.

3.2.2 Trade Secret Law

Trade secrets are covered by “contract law”. Basically any kind of information which is of (economical) value for a company or any other trading institution can be
protected by a trade secret. R&D\textsuperscript{26} data, marketing strategy information, customer lists, and proprietary techniques or formulas can fall under a trade secret protection.

Company X wants to be the first competitor at the market with a new product. Cooperation with other non-competitive companies might be useful and necessary to reach that goal. The contract between these companies might be covered by a trade secret protection. The terms of condition of such an agreement can forbid, for example, any other negotiations which other competitors of the company X. The value for the company is the idea for the new product. This idea cannot be protected by other means.

In the software industry it is a common term of contracts with employees that they are not allowed to talk about their work in the company to external persons. This is of course an issue for debate\textsuperscript{27}.

"Those who criticize trade secrets laws maintain that they are often used to restrict employee mobility. In addition, they argue, companies cannot claim to own what's in an employee’s head." (Spinello, 1995, p. 150) This might be a valid criticism but the employee knows about the "terms of condition" in the contract before hand. Let us take an example where an employee changes her employer. To be precise one has to point out that the employee owns his mind and is autonomous in his actions. On the other hand, the new employer has a right to take advantage of the new employee. From a moral point of view I would argue that the employee should remain silent about the business idea of her former company. I know that this is a very unpleasant situation for the employee. That is why the criticism put forward by Spinello has some force.

A more general problem of trade secrets is the dependency on confidentiality. Again, Spinello states "[t]he primary problem with relying on trade secrecy protection is that if the trade secret somehow becomes public, either intentionally or accidentally, it loses its proprietary status and becomes part of the public domain, so there is no way to stop other [software] developers from using it." (Ibid, p. 151) In general it is very hard to tell where and when and by whom a trade secret was betrayed.

It should also be mentioned that trade secrets do not exist in the open source world. Most often projects publish a ToDo-list where they state which kinds of steps the project will take in the next development period.

\textsuperscript{26}="R&D" is a commonly known shortcut for "Research & Development".

\textsuperscript{27}I made up here an example with has relation to my topic. A very obvious example for trade secrets is the Coca-Cola recipe. If someone would get hold of this recipe the future of the company would be threatened.
3.2.3 Patent Law

“A patent is usually seen as an agreement where the state gives a monopoly for a certain period to the inventor who gives technical information to the public.”\(^\text{28}\) for a particular time duration; which is normally 17 years. The main distinction in comparison to a trade secret is the disclosure of a patent. A patent is only effective and valid if it is published and competitors and future engineers can study the description of the technical invention for which the patent was granted. Mathematical algorithm, law of nature, non-novel and non-obvious inventions, methods of doing business cannot be covered by patent law (cf. Hettinger, 1989, p. 32f).

“Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent” (Spinello, 1995, p. 151) This view has to be alleviated. “What one patents are inventions – that is, processes, machines, manufactures ...” but “specifying what sorts of 'technological recipes for production' constitute patentable subject matter involves distinguishing specific applications and utilizations from the underlying unpatentable general principles.” (Hettinger, 1989, p. 33) So, it is not sufficient to claim an idea in a manufacturing process instead it has to be applied in a “particular utilization”. The question which matters here most is if software programs fall under this definition? Software programs are mathematical algorithms translated into a programming language.

Now, if one considers computer calculations (basically mathematical algorithm) as a part of the process [e.g., a machinery], then they might be patentable in this particular utilization.

As a result of this decision, more and more companies are seeking this viable form of intellectual property protection. (cf. Spinello, 1995, p. 151f)

The risk here rests in the implementation of a mathematical algorithm and a process. The process might be held as the software as such, while a algorithm is only a part of it. In practice than it is hard to decide if the algorithms in a software are patentable or not. Another difficulty in practice is to figure out where and when a real invention had been created.

However, “more and more companies are seeking this viable form[patent] of intellectual property protection.” (Ibid., p. 152)

3.2.4 Copyright Law

Another form of protection for software programs and products is copyright law.

\(^{28}\)http://www.prv.se/eng/patent/index.html, accessed 05/05/2004
The Berne Convention for the Protection of Literary and Artistic Work formulated the foundation for the protection of literary and artistic work for the author. I think it is sufficient to refer to this foundation of copyright law. The copyright which is applied in the legal system.

In article two of the convention we find the following formulation.

(1) The expression "literary and artistic works" shall include every production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression, such as books, pamphlets and other writings; lectures, addresses, sermons and other works of the same nature; dramatic or dramatico-musical works; choreographic works and entertainments in dumb show; musical compositions with or without words; cinematographic works to which are assimilated works expressed by a process analogous to cinematography; works of drawing, painting, architecture, sculpture, engraving and lithography; photographic works to which are assimilated works expressed by a process analogous to photography; works of applied art; illustrations, maps, plans, sketches and three-dimensional works relative to geography, topography, architecture or science.

(2) It shall, however, be a matter for legislation in the countries of the Union to prescribe that works in general or any specified categories of works shall not be protected unless they have been fixed in some material form.29

Since everything which is expressed can be art according to the quotation above, the obtaining of copyrights is easier than the obtaining of patents. Copyrights, for instance, do not have to prove the novelty of an [artistic] invention.

"Copyrights give their owners the right to reproduce, to prepare derivative works from, to distribute copies of, and to publicly perform or display the "original work of authorship." (Hettinger, 1989, p. 34) The question which matters here is how this declaration-article can be applied to the protection of software programs. "The original written source code is the authorship, the program (consisting of the logic and design of the software) is the expression, and all forms of software (written, printed, ROM, or diskette) from which a version of the program can be produced or communicated with the aid of a machine or device are protectable copies." (Spinello, 1995, p. 153)

Thus copyright infringements could occur in many different ways. The software is protected by a legal recognized and enforceable protection, the copyright. The terms of condition of a copyright classify how the software is protected and what the user is allowed and not allowed to do. Spinello, for instance, argued that "an unscrupulous developer could blatantly copy lines of source code and sell them as his..."
own program.” (1995, p. 153) This presupposes that the original author forbade the copying of the source course. Spinellos reasoning after taking up this example is sound but it is based on quite limited presuppositions. However, the author of a software is entitled to restrict and provide a statement of requirements for the usage of the piece of work she labored on.

The copyright is defined by the terms of the condition in a license. This license might for instance require to purchase a extra license for every copy of the software. On the other hand, as we have seen in the first part of this chapter the copying process can be performed at almost no cost. The protection of the authorship by a copyright is ensured for the lifetime of the author plus 50 years after his death. In an ethical analyses one has to ask what “protection” hear should be interpreted? A sound justification for the protection of the work is the incentive for a (monetary) reward of her work. Companies, on the other hand, do not “die”.

3.2.5 Concluding Remarks on Intellectual Property Rights

As we have seen intellectual property rights, as well as “normal” property rights, are rights of exclusion. Exclusion here is understood as avoiding other people to get into the protected sphere. A right to use intellectual property can be obtained for a monetary reward in return. At least this is the practice for almost every form of intellectual property. The obtaining of a license, for instance, is the most often used practice for gaining access to protected goods, objects etc. Intellectual property rights were made and are based on the assumption that exclusive rights would enhance innovation. This is an one-sided utilitarian approach. In the last chapter of this theses I will question the conclusions which had been drawn from this utilitarian approach.
4 Ethical Aspects of Open Source Software Development

In order to fully understand the claims by a movement which is somehow “invisible” I thought it is useful to present some arguments from the grass-roots. This empirical material does not meet the requirements to be representative but it offers a useful inside view on the way open-source software engineers think. I will first presents some arguments from the “bible” of OSS: Eric S. Raymonds the Cathedral and the Bazaar. This book is a collection of articles by Raymond which he wrote throughout the years as he worked as a software engineer at RedHat Inc. and is the founder of the fetchmail

Raymond is one of the people in the OSS movement which also wrote about the free software world and the culture which we find there. I think it is necessary, in order to understand the mechanisms of the engineering process to study for a while the voices from the grass roots.

I will present and explain here some arguments from people involved in OSS engineering which I collected from a maillinglist at the Free Software Foundation Europe. I am in great debt to these people because they explained very careful their culture and ways of understanding to me. On the other hand I present here, as I said, some arguments from writings which had been done here and there by people from the movement. I will not distinguish here among the different “clans” in the open source world and their particular arguments. As many cultures, so is the open source culture an incoherent group with different views and values on and to different things. My purpose here is to present and distill arguments which are strong claims for or against the application of the term “property” to software. And these claims emerge out of the whole movement. (I know that some people do not like the term movement)

In the first part of this chapter I will present some argument from “homesteading the noosphere”, a quite important chapter, as I would say, by Raymond in his the

Cathedral and the Bazaar. Then I will continue to present some arguments from a very recent discussion at the Free Software Foundation Europe “discussion” mailing list https://mail.fsfeurope.org/mailman/listinfo/discussion.

If the reader is not familiar with the many distinctions of software or the term as such I recommend to read first the third chapter of this thesis. In the third chapter I am going to say more about the nature of software and the intellectual property rights as say are applied today.

On of the major distinction in software is the one between open source and closed source software. The term open source refers to the human readable “text” which constitutes the software and which is written by the engineer\(^{31}\). This “text” can be studied and changed by a human being. Improvements, simple changes are easy to perform for advanced engineers.

In order to make use of this code with a computer it has to be compiled. A compilation is necessary in order to execute the code on a computer. The compilation is performed by another software – the compiler. There are many such compilers available. What also matters in this case, is the “language” in which the software is programmed. A program, very briefly, is a special form of a mathematical algorithm which is made for the computer. “Broadly-defined, an algorithm is an interpretable, finite set of instructions for dealing with contingencies and accomplishing some task which can be anything that has a recognizable end-state, end-point, or result for all inputs. ... Algorithms often have steps that repeat (iterate) or require decisions (logic and comparison) until the task is completed.”\(^{32}\). The terms and signs which are used to express the algorithm are defined by the programming language which is used (e.g., JAVA, C, C#). The coherent set of commands and instructions is necessary to translate the algorithm into the programming language. By compilation this programming language is translated into a computer language which can be used by the computer to perform a task. The computer language is NOT human readable. It is possible to guess what certain things might mean but in a programm of 3 millions lines of compiled code it is practically impossible for a human being to read or understand the text. This part if the software is described as closed code. Proprietary software is basically closed code. The open code (in the programming language) is not available for the public but hidden and protected by the company.

Now, what matters here is the source code; the literal text of the program. Software engineering means basically developing the source code. The compilation of the

\(^{31}\)Actually, it is also possible that another software “writes” or produces software.

\(^{32}\)http://en.wikipedia.org/wiki/Algorithm, accessed 05/05/2004
source code is full automated (by the compiler). As we will see here, the development process has its own “laws”. Laws which we are as scientist quite familiar with: motivation, reputation, mutual respect etc.

Before I turn to the arguments from the open-source side I would like to mention some arguments for closed (or proprietary) software engineering. Closed code distribution has become a business model in the 80th of the last century. The distribution of closed code together with a restrictive license which only allows the usage of the software for a limited number of computers at the same time became a new market for the emerging software industry. The distribution of closed-code-only allowed them to protect their intellectual property. I may shortly mention here that this is an artificial protection of software; it is against the professional understanding of behaving and working in the professional field of software engineering. The nature of the profession explicitly allows and wishes the distribution of source code. But more about this issue will be presented in the next chapter.

However, closed code defenders argue that their source code contains millions of lines and that no one would be really interested in investigating this code. Additionally, an access to the source code and a change of the source code followed by a new compilation would abolish all guarantees for the warranty of the software. (cf. Grassmuck, 2002, p. 234f) On the other hand, proponents of the open source code world argue that no software does 100% what the user expected the software to do. So equipped with the necessary know how this software can be improved without any major obstacles – if it was free open source software.

Grassmuck (2002) mention in his book about free software also another disadvantage which become a major reason for some public administration to “migrate” to free software. Grassmuck said, that “modifications on proprietary software are technically hard to realize and according to the the terms of the license forbidden. The formats of the data the software writes [e.g., a saved document in the .doc format, A.S.] are very often not documented.” A major change in the software or the end of the production of the software might cause a loss of data which is saved in such a documentation of a data format contains a detailed description in which way the data is stored in a file. For example this expression “<text:span text:style-name="T2">EXAMPLE</text:span>” will make the word “EXAMPLE” appear in bold letters in the word processor OpenOffice. This is predictable because the format of the data-file is well documented and free available. Some of these formats are standardized (like XML (OpenOffice) or HTML (see also: Standards and Standards, Ch. 2) and the documentation is public available. All open source standards are public available, some proprietary are not. A free available documentation of the OpenOffice file format is available at http://xml.openoffice.org/ (accessed, 28th of April, 2004)
documents. That is why I hold: to ensure the freedom to modify software and maintaining it will be one of the major tasks of the legislation and public administrations in the future. This refers to the argument I mentioned in the 2nd chapter about the importance of software in the public sector and for the citizens.

Now, after clarifying a little what open source software is and what the main difference in comparison to the closed code model of engineering are I would like to present, finally, some arguments or inside views from the grass-roots. The grass-roots constitute itself out of people who participate in the development of software or the usage of such software.

### 4.1 The Cathedral and the Bazaar

At the beginning of his famous book Eric S. Raymond said that

> “[t]he essays in this book did not invent such a fundamental advance, but they do describe one: open-source software, the process of systematically harnessing open development and decentralized peer review to lower costs and improve software quality. Open-source software is not a new idea (its traditions go back to the beginning of the Internet thirty years ago), but only recently have technical and market forces converged to draw it out of a niche role.”

These few sentences contain almost all arguments which are put forward in favor of open source development: peer review, low cost and improved software quality. This way of developing is possible because of the “growth of the internet and its wide spread used” at the beginning of the 1990th. The internet was developed and used so far by scientist which where “insiders” in network technologies. Now, the internet became through a “public discovery” available to (in theory) everyone. It became a mass medium. First of all university students made use of this new medium. So in 1991 “a Helsinki University student named Linus Torvalds ... began developing a free Unix kernel for 386 machines” (Ibid., p. 15) But Linus Torvalds did not decide to do it on his own. Instead, he published the source code on the internet and asked people to help him. He also picked a license, the GNU General Public License, which ensured every co-developer free access and usage to the joint-venture developed software. This was the big change and the start of the successful history of open source. What Linus Torvalds did was nothing outstanding or technically advanced – it was about using the culture of programming and unifying it on/over the internet. As I said earlier – open source software engineering is only possible with an electronic network like the internet.

What emerged with this way of programming was labeled later on by Raymond
the Bazaar”. The Bazaar refers to the way the development of a particular software proceeds.

Raymond, a well experienced software engineer said:

“I believed that the most important software needed to be built like cathedrals, carefully crafted by individual wizards or small bands of mages working in splendid isolation, with no beta to be released before its time. […] Linus Torvalds style of development — release early and often, delegate everything you can, be open to the point of promiscuity — came as a surprise. No quiet, reverent cathedral-building here — rather, the Linux community seemed to resemble a great babbling bazaar of differing agendas and approaches” (Ibid., p. 21f)

This really was a surprise but it was, in fact, nothing new. Gerald Weinberg referred in his book about the Psychology of Computer Programming to the spirit of sharing and peer reviewing during the development process. What was new is the fact that there is hardly one goal which had be reached when the software is finished — there are many. The cathedral model of software engineering refers to the structure of the technical process and also to the social environment of the development group. Since the emerge of the Bazaar style successful software development was held to be dependent on the following factors: definition of the goals, monitoring the development, motivating the developers, organizing the deployment of the people for best productivity, and to marshal resources needed to sustain the project. (cf., Ibid., 57) The definition of the goals was held to be important to place the new product successful at the market. The other points were held to be a necessary condition in order to reach the best possible result. The “Bazaar” was hostile to such definitions. But it worked anyway. Why, should be explained now.

Raymond suggested in his book nineteen points for a successful Bazaar project. I will refer to some of them but will keep the numbers in accordance to the book. Additionally I will give some comments on these points. The purpose is to demonstrate which are the real advantages and obstacles in open source engineering and what are the advantages compared to closed code software. It is important to keep in mind that these points are more or less describing a way of writing software where software is free and open source, which means is not restricted by intellectual property rights.

1. [Every good work of software starts by scratching a developer’s personal itch.]

The idea behind is sentence is that software does not necessarily need a market — a necessity is almost sufficient. ‘Necessity is the mother of invention’ is a quite often quoted slogan in the open source world. Very often, a simple program is needed

34 All these point can be found in The Cathedral and The Bazaar from page 23 onwards.
to fix a simple problem. This software, on the other hand, might become one day a valuable part of another software. Or: the one and the same program will be constantly improved and “find” more and more developers which gets attracted to it.

2. [Good programmers now what to write. Great programmers know what to rewrite (and reuse)]

The problems programmers face are mostly not new. But it could be that in another environment (e.g., another operating system) or for a slightly different purpose a “new” software is needed. Raymond wants to say here that the smartest solution is to look for a software which is almost fits to the desired solution. The next step then would be to rewrite this piece of software.

3. [Plan to throw one away, you will, anyhow. Or, put it another way, you often don’t really understand the problem until after the first time you implement a solution. The second time, maybe you know enough to do it right. So if you want to get it right, be ready to start over at least once.]

This is something every scientist is probably familiar with. A draft of an essay or a first outline of a great idea may fail due to the critical review of peer students or colleagues. But it might also fail because you do not hit the core by making imprecise formulations and so on. In the open source world these wrong tracks are immediately discovered by the community.

5. [When you lose interest in a program, your last duty to it is to hand it off, to a competent successor.]

This is probably one of the most advantages of the open source world. Orphaned projects which are not maintained any longer can be captured by an interested individual or group of people which are willing to continue the software project. It is important to not that such an “adoption” has to follow a unwritten law. Potential interest in a “dead” project has to be announced at well known maillinglist and usenets. The old author or project group has to be informed about the intention to adopt the project. Usually the change of the main author or project group will be officially announced through the media mentioned above.
6. [Treating your users as co-developers is your least-hassle route to rapid code improvement and effective debugging.]\(^{35}\)

7. [Release early. Release often. And listen to your customers.]

Users play a very important role in open source software development. People which will actually use the software in everyday life situations are considered as the most important status group within the software development. Every software has some minor mistakes (bugs) when it is release the first time to the public. But to minimize the number of bugs in a software is one of the most desirable goals of every software project. So, many versions of the software are released. Often and early. Often refers to the number of releases. Early refers to the development process. If the project group agrees that a new step or a major advancement in the software had been implemented an early release (right after the implementation) will show if it was successful or if it causes serious problems. Users have the very important task to test this software and report the bugs or mistakes they experienced while using the software back to the development group.

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

“Given enough eyeballs, all bugs are shallow” - this slogan is also known as “Linus [Torvalds] law”. The basic idea here is that a constantly peer review of parts of a computer program will eliminate almost every bug. Additionally, the bug-report of the users (beta-testers) will indicate the order in which the bugs will be fixed. It should also be said here that many eyeballs and many imprecise bug-reports from the beta-testers do not make work for the developers easy. In practice many bug-reports from users have to be critically reviewed and evaluated. I think it is fair to say that this way of software developing works but it is for sure not the best-possible one.

9. If you treat your beta-testers as if they’re your most valuable resource, they will respond by becoming your most valuable resource.

\(^{35}\)“code” refers to the program as such. “Debugging” refers to the elimination of mistakes in the software code. A funny story why the word “bug” is used can be found here: [http://en.wikipedia.org/wiki/Computer_bug](http://en.wikipedia.org/wiki/Computer_bug) (accessed, 04/04/2004)
If you treat people like persons, they will become persons. Mutual respect in every sphere and appreciation of what someone is doing is one of the high values the community put forward. If developers would not appreciate the input from the users the software will take significantly a fast and high-quality development process.

14. Any tool should be useful in the expected way, but a truly great tool lends itself to uses you never expected.

As I said earlier, some software has capacities which the original developer is almost unaware of. A continuously improved software which fulfills basic task of computing will become very useful and do a great job as a part of another software. Software can also be turned in another direction.

4.1.1 The “Community”

The “community” is a very vague term but still the most often used in the open source world and public debates. The community captures a number of people and professions which are somehow related to the development of software, like university students, administrators, beta-tester, teacher, school students and many other people with an interest in software and open source software development.

Open source project are more “narrow” organized. An open source project can be composed of one or more people. The number of developers may also vary but as long as the software is maintained at least one developer exist.

It should also be noted here that companies develop open source software. Some years ago this was a special case. Nowadays it is getting more and more usual. Why this development emerged will be explained later but it should be clear that valuable open source software projects are not any longer sparetime activities by some altruistic people – it gets a serious business.

Anyway, the main task for a founder of a new project is to find people who are willing to join the project (community building):

“When you start community-building, what you need to be able to present is a plausible promise. Your program doesn’t have to work particularly well. It can be crude, buggy, incomplete, and poorly documented. What it must not fail to do is (a) run, and (b) convince potential co-developers that it can be evolved into something really neat in the foreseeable future.” (Ibid., p. 47)

This really seems to be the secret for attracting people for your project: having a plausible promise (for success). But these are only the start conditions of the project. The maintaining of the project group is almost more important since this will guarantee the long-term success of the project.
4.1.2 The Social Context of Open Source

As we have seen so far many factors play an important role for a successful open source software development. What in the literature written by advocates of the open source model or by “independent” software engineers like Weinberg (1998) is on and on mentioned is the virtuous characters of the project participants. This is a necessity to lead the project to a success. Raymond put it this way:

“The developer who uses his or her own brain in a closed project is going to fall behind the developer who knows how to create an open, evolutionary context in which feedback exploring the design space, code contributions, bug-spotting, and other improvements come from hundreds (perhaps thousands) of people.” (Raymond, 2001, p. 51)

The virtuous behavior find their expression in the principle of common understanding. As opposed to the principle of command and discipline. The point of project leading is not to force people to stay in the project but to create an atmosphere of mutual respect and learning.

“I think that the cutting edge of open-source software will belong to people who start from individual vision and brilliance, then amplify it through the effective construction of voluntary communities of interest.” (Ibid., p. 54 [emphasized by A.S.])

These views are expressing somehow the atmosphere and spirit of open source projects. I think it is easy to see that artificial compositions of high skilled software developers lack some of the major capacity of an open source project. The “cathedral” software development as such lacks the capacity of letting the people choose in which way the development process will go. The main reason for this is that the software which is developed has to match the market requirements because it is itself an end, software is the main product of the company. In the open source world the software is more or less considered as a means for the abolition of a problem. This conceptual distinction has serious consequences for the application of intellectual property rights to the software. So, software is not equal to software – and this holds fin many respects.

“Perhaps in the end the open-source culture will triumph not because cooperation is morally right or software “hoarding” is morally wrong (assuming you believe the latter, which neither Linus nor I [E.S. Raymond] do, but simply because the closed-source world cannot win an evolutionary arms race with open-source communities that can put orders of magnitude more skilled time into a problem.” (Ibid., p. 54)
4.2 Voices from the Grass-Roots

So far I presented a summary of the community architecture and why open source engineering works. As I said already the “community” is a divergent organization with many different views. Now, I would like to present some arguments I gathered from a discussion on a public maillinglist. The archive of the discussion can be accessed here: http://mail.fsfeurope.org/pipermail/discussion/2004-April/subject.html#start, the subject is “Ownership in Software”. The “archive” contains all messages which has been send to the mailinglist in a particular period of time. In this case the period is one month.

This maillinglist is a platform for general discussions on and about free software. The list is hosted by the Free Software Foundation Europe. Maillinglist are frequently often used tools to discuss issues, to ask for advices, to publish announcements, or list which are in another language than english. This particular discussion list I used here is not so much concerned with particular software projects but with general discussion about free software. It should also be noticed that a subscription to a particular maillinglist allows every member to send an e-mail to an email-address. This email than will be automatically delivered to all subscribers of the list.

I followed so far many discussions on this list quite passively. This time I raised a question by myself. I asked why software should not have owners. The following discussion on this list was the very interesting and I am in deep debt to all the people on the list who contributed with their valuable comments to this discussion.

The presentation of the arguments is necessary as I would say because they offer an inside view on the “internal” arguments which come from specialist which work in their everyday life with free open source software. I also hope to be able to present some forceful argument against the applicability of intellectual property right at is done today.

I decided to present the main arguments in a logical way. I will give comments to every set of arguments. I only present here arguments which I hold to be meaningful for my purposes here. I encourage the reader to validate the arguments I presented here – like in the spirit of open source. (Please, do also note that I tried to improved the orthography for the presentation here. Maillinglist discussions are most often quite energetic.) I tried to classify the arguments brought up during the discussion. I will start with “Freedom/Power”
4.2.1 Freedom and Power

I will start with my incentive argument which dealt with the freedom to claim rights in software. I claimed that the possibility to claim property rights in software should be the freedom of every software developer.

One of the responded wrote "because to satisfy that "freedom" for a few individuals you restrict freedom for society as a whole, for instance? Because it is evil to act your power in spite of others?" This argument shows that there is a huge sensitivity for the right balance of claims. On the one hand claims of individuals and groups as producers of software and on the other claims of the "society" which has somehow to benefit from the software. I hold that this might become a key issue. Claims of the developers are nowadays realized through a system of intellectual property rights. I still this term because I did not develop a better set of concept of the justified claims so far.

"For a couple of centuries we’ve accepted that there’s no ownership on literary works. Could you [mean is A.S.] tell me why do you think that’s wrong? I can see no strong argument for ownership of software. You say we need good software, and you claim that’s possible by limiting our ability to correct bugs and to maintain a program beyond its creators wishes to fit our purposes. That’s seems strange to me. You [A.S.] say it’s a matter of justice, freedom and valid claims. I completely disagree. I see no justice nor freedom in being restricted to do something by others. I don’t believe any restrictions on my freedom to do things to be valid claims without a great explanation. I hope you have one." (João Miguel Neves, 04/21/2004)

Here we have another argument. The possibility to “fix” a problem in a software is valued as a positive freedom. In practice, administrators of computer networks have to use a particular kind of software which is proprietary. That means it is usually not allowed to change the software for the purpose at stake. From this point of view the argument is sound. But it loses a bit of its force when we take another example. Sometimes it is possible to use another software, an free open source software, which does the same task. So, it is not under any circumstances necessary to have access to the source code and the “fix”-argument has to be evaluated carefully.

4.2.2 Copyright / Authorship / Ownership

The most interesting part of the discussion was about initial question of ownership in software. One participant claimed, “[t]he owner is the society, that educated the engineer, that gave him opportunity, that gave his job and life. The inventor has always a great DEBT to the society, because he had access to the information,”

36 http://mail.fsfeurope.org/pipermail/discussion/2004-April/004416.html
but a lot of others didn’t. Guys, do you have access to the internet and LOTS of
information, but in my country 50% are below the poor line. Don’t you think you
have a big debt? How can you pay all the information you have? Giving it back!”
(Ricardo Andere de Mello, 04/21/2004) This is one radical argument which only
a small force, as I would say. I hold that the society can not and should be the
owner of software. This argument denies the social institution of private property
and is that’s why a little to radical. But the second part of the argument is not
without any force. The development (and usage) of software is a global issue. Rights
of any kind, e.g., through software patents, granted in one country may not have
any force in another one. The claim here (“you have a big debt”) is pointing, as
I interpret it, to a restrictive world wide intellectual property law which makes it
hardly possible for third world countries to take part in the information society. A
world-wide application patents to software on a global scale would hinder the world
wide development of open source software. Since patents are only applicable to new
inventions and software is basically the translation of algorithms into a programming
language it be impossible to grant patens to software. Algorithm are laws of nature.
Laws off nature are by definition not an object for patent protection.
The following argument refers to a broader problem which I will take up later (->
commons and anti-commons). “Proprietary Software can cost lives, it’s just like
patents on medicines. Do you know a company finished genoma project three years
earlier and sell the genoma during these three years? Can you imagine how many
medicines would be invented in these three years, and how many lifes vanished just
because a company’s ambition?” (Ricardo Andere de Mello, 04/21/2004) I do not
know and I did not validate proprietary software cost life but the expressed threat
that “patents causes lifes” has some forces (see: commons and anti-commons).

COPYRIGHT

Now let us turn to the question of copyright in software. What a copyright is will
be explained in the next following chapters.

“Copyright is only a special privilege granted to the way one expresses
information, not the information itself. Copyright wasn’t designed as a
property right, quite self-consciously because it participates in the realm
of information. Copyright was designed to foster the greater dissemination
of useful information by granting authors the privilege of being paid
for each static copy of their original expression that’s produced. Informa-
tion production is a good bit different from other forms, particularly
in the kind of product that’s produced. Ideality is not the sort of thing
that can be "owned" in the first place, because it’s inherently shared as
soon as it’s communicated – and when it comes to logic and algorithms,
you’re definitely comfortably in the realm of ideality.” (Seth Johnson,
04/21/2004)
I do not really agree with this line of argument. I do not agree with the premisses. Some people in the community state that software is basically information and that information can not be owned. Hence the application of a protective right to software is unacceptable to them. This argument has some weaknesses. I think it it wrong to consider software just as a kind of information. A software with 3 millions lines of code cannot be considered as just a bundle of information. Software is also executable and the sequence of the commands has to follow the rules of the programming language. That is why I hold that the claim is a underestimating the whole quality of software and is that’s why inapplicable. If we accept this than it should also be allowed to say that software is also an expression of an idea.

Information can not be owned – that is true but as we have seen and will see software is much more than that. That is why a copyright protection for the labor on software can be applied.

“Software is product of the mind, is Information! You take some parts of information, process them in your brain, and then combine in a new form and exposes it as software. At the end, like lavoisier, nothing is invented, everything is transformed. Duchamp with his shit cans showed that everything can be art. I talked to Stallman sometimes I can guarantee you Stallman knows GPL is a free software "defense mechanism", not the main idea.” (Ricardo Andere de Mello, 04/22/2004)

Now the question is how restrictive this protection should be. And here I want to emphasize a very important point in respect to the copyright protection of software. It should be noted that free open source software is protected nowadays through copyrights. So, a copyright is just a tool to protect the work of the laborer. Later we will see that this tool has much more qualities than just to “protect” the labor. The author is the one who is allowed to formulate the terms of agreement for the license which constitutes the legal instance of the abstract copyright. Any use of the software which violates the terms of the conditions in the license is a violation of the license, hence a violation of the copyright.

The ethical question here is how far should a protection of an intellectual piece of work reach. What is necessary and beneficial for the author and for society? What claims are valid and how should they weight against each other?

A question which is highly debatable than is if software should have owners. (cf. Johnson, 2001, p. 140). The term “owner” is very strong in relation to software. Here we have a good argument why it is hardly applicable to software:

“So yes, we all agree you are granted authorship, what we disagree with is exclusive and strict ownership. If you are the owner of your work, it means you are also the owner of __my__ copy of your work. (Here "copy" means "instance" or "specimen").” (Alessandro Rubini, 04/21/2004 [emphasis by A.S.] )
Ownership is different than authorship. Here we have a very good example how to distinguish these both:

“Forget a moment about software. If I take a piece of wood and make out something of it. I’m the author of that piece of art *and* the owner of the piece of wood. If I sell the piece of art I created I’m not anymore the owner, but still the author. Right? Ok so authorship and ownership are 2 very different things that applies to many things ... .” (Simo Sorce, 04/22/2004)

This argument is very sound and applies first of all to material goods. Clothes designed by a famous couturier is considered as a piece of art. The author or “the artist” owns the authorship in the design of the clothes but the costumer owns the piece of clothes which had been turned into art by the couturier. Here the distinction is easy to make. Every unauthorized reproduction of the clothes will violate the rights of the author of the design. The design is a valuable part for the production. The artist spend many hours to invent the design. She should get a compensation in return because the real value of the clothes is founded through the design.

When it comes to software it is a little different. Software has to be useful. It has to work properly. The claim of users to hold the software company warrant is based on the functionality or miss-functionality of the software. The functional design as protected through the copyright has a slightly other meaning here. It is the elementary purpose of the software to function in a predicted way while art in the original sense has not to “work”. It is embedded in the dual nature, which I mentioned already, that make the non-applicability of “ownership” to software a little difficult. The labor of the author in software would be protected by a copyright but since software is an intangible good authorship and ownership are bounded together. The re-selling of a piece of wood with the “art” embedded is no real problem for the other. But computer software can be copied without loses. That means the the piece of art and the piece of wood are produced at the same time. To protect the labor of the artist (software-developer) it is useful to use the term authorship. Software on the other hand is also a “product” in the technological sense. It is not only the developer who is involved in the production and distribution (selling) of the software. That is why the only application of authorship and the denying of ownership to software is difficult to justify.

4.2.3 Software is Speech

Another interesting argument was used in relation to the programming language computer engineers use when they build the software. A programming language has a “grammar”, a “vocabulary” and an “orthography”. Under grammar we may
understand in which order the commands have to appear in the software. A very often used commend in almost every programming language is the “IF ... ELSE” clause. Here it is defined that a first “IF” has to appear and then “ELSE”. Have a look a this small example written in the scripting language PHP:

```php
<?
if ($house=='red') {
$coun try="The coun try has to be Sweden! ;-)"
}
else {
$coun try="The coun try is probably not Sweden. :(";
}
echo $coun try;
?>
```

If you exchange “IF” and “ELSE” then the software wont work. So too know the grammar is very important. It is also important to know the orthography and the “vocabulary” (here: the list of commands) which is available.

Some members of the community argued that “speaking” the language is a matter for the right to free speech and not of ownership in software. “If you [A.S.] permit ownership of software, you permit ownership of language. Computer programming languages are a language understood by computer programmers, who usually write it fluently enough to create works. If you restrict and allow ownership of software, you effectively restrict programmers’ freedom of speech ... ” (Rui Miguel Seabra, 04/22/2004)

I think in a certain sense this argument is sound. I denies strictly for example the possibility to apply patents to software. I think this is the only argument which can be derived against restrictive property rights. I do not think, on the other hand, that argument is sound when we use it in order to defend authorship in a software. Every person can use the language irrespective of the use by others – just like ordinary languages. The claim that “ownership” in speech would restrict the programmers “freedom of speech” is besides the point. The freedom of speech guarantees the freedom of expression by almost any legal means. The copying of authored work is, on the other hand, not a matter of freedom of speech.

But this argument has also another side. Some simple programs would be build in nearly the same way in similar programming languages. The risk that a software would violate protected code in these cases is extremely difficult to judge. In such a

---

You can try this example out here [http://newedu.de/thesis/example.php?house=red](http://newedu.de/thesis/example.php?house=red). Just change the word “red” at the end of the URL and you will see what happens.
case the expression of “freedom to speech” has some force. However, best understood can be the “freedom of speech” problem by this quotation I think.

“Computer programs are writings and the protection of the exclusive rights to these writings is made via the Berne convention. Free speech can use any way of expression. Computer programs is a way of expression. "Expression" of something is covered by the Berne convention. The author have the rights to apply these exclusive rights in the form he wants[*]. (from making copylefted Free Software to make proprietary software or confidential software)” (Alexandre Dulaunoy, 04/22/2004)

4.2.4 Final Remarks on the Voices from the Grass-Roots

Finally I would like to make some final remarks about the problems computer engineers and society faces when they work with proprietary software or when they use it. This might lead us to found a sufficient argument why restrictive property rights in software should be thought through.

An participant presented at the end of the discussion this list with problems which are related to the narrow field of software engineering but also to to public issues like accessibility of ones own data, etc. He wrote that an engineer should have:

- [a] The right to fix problems (especially if the "owner" doesn’t do it).
- [b] The right to get compatibility (if interfaces and data formats are undocumented, sometimes this requires reverse engineering)\(^{38}\).
- [c] More generally and related to the previous point, the right to your own data. (If you store them in a proprietary format, you might not be able to do with them what you want, or you might even depend on the proprietary vendor (repeated license payments etc.) just to be able to access your own data on another computer, or in the future, ...)

- [d] The right to check if the software does anything nasty which could harm you (intentionally as in backdoors, or unintentionally as in security holes). This requires source code or reverse engineering.
- [e] The right to your computer at all. (Think about DRM, or about required remote access to the "owner" for checking your license compliance, etc.)
- [f] The right to do unrelated things. (I think some proprietary companies once tried to restrict you from using certain free programs at all on the same machine. I don’t know, though, if this restriction was legally valid.)

(Frank Heckenbach, 04/22/2004)

In short, this quotation summarizes most of the problems with proprietary software. It also points to the difficulties of the contemporary concept of intellectual property rights.

Earlier in this thesis I presented some possible categorizations for software. I think this chapter provided a much more sufficient view on how software is developed in on the internet in the open source way.

---

\(^{38}\) It should be mentioned here that “reverse engineering” is a method to generate the source code out of a binary file. This procedure is by most license of proprietary software forbidden.
I hold that this development carries its own ethics and should therefore strongly be taken into consideration about the usefulness of the application of restrictive intellectual property rights, such as patents, to software. The major reason why a society grants those right is to protect the author from an exploitation of his investments. The open source demonstrates that this assumption about software engineering is not valid any more and lost that’s why its force.
5 Consequences of Ownership and The Value of Virtues

The term property expresses a legal entitlement to an object. The owner of this object is also entitled to a set of rights which he may use to enforce or practice her ownership. He may use, destroy, donate etc. his property.

The effects of ownership or non-ownership of object for the society vary very much with the object which is owned. To own something is to fence it in some ways. A garden can be fenced and books are "fenced". But some things simply lack the capacity to get fenced. The Seven Seas cannot be fenced, air cannot be fenced. The idea of intellectual object can be fenced is widely accepted, but can they? To what extent does it matter how important the object is for society? Earlier, I mentioned in a footnote the change of the meaning of water when it became scarce. How should the society sets the rules of ownership in properties of object which were free and non-scarce, but became scarce through the course of time.

This chapter will present theories about the consequences of exclusive property rights in objects. It will also present a reasonable explanation why, in some fields, property rights should not be applied. Scientific work and knowledge, for example, should remain free in order to remain scientific, as one might argue.

5.1 Commons and Anticommons

As I said in the introduction of this chapter I will be mainly concerned with the consequences of property here. Now, I would like to introduce some ideas about the need or non-need of property (or any other form of regulative means) and the consequences for the commons. All legal rights are socially established – even though they need a philosophical foundation; they have to be justifiable. A right, once established, mirrors the morality, values and the conscience of its society. It also mirrors the understanding or conceptualization of social institutions, circumstances and goods. Rights, basically, are applied to these social institutions and goods. The
justification of rights to/in particular circumstances may change during the course of time. This might have many reasons. But the once well-working right or concept is historically and object depended; that is why it might be re-evaluated or re-judged.

5.1.1 Hardin’s “The Tragedy of the Commons”

One of the empirical reasoning about the need for property rights was developed and exemplified in the so-called tragedy of the commons by Garrett Hardin. Hardin (1968) introduced this idea in an influential article in the Science magazine. His main argument was that the shared use of common resources will lower the usefulness for all people. If there is no restriction, e.g. a private property right, for the use of common resources these resources will be misused or/and its value for all people will decrease.

Adam Smith in his The Wealth of Nations popularized the 'invisible hand,' the idea that an individual who 'intends only his own gain,' is, as it were, 'led by an invisible hand to promote ... the public interest'.” (Smith, 1937, In: Hardin, 1968, p. 1244) Hardin goes on in his argumentation and concludes that “If the assumption [that men will control their individual fecundity so as to produce the optimum population] is not correct, we need to reexamine our individual freedoms to see which ones are defensible.” (Ibid., p. 1244) This is a very interesting point to which I will turn back later on. “The day when the long-desired goal of social stability becomes a reality. At this point, the inherent logic of the commons remorselessly generates tragedy.” (Ibid., p. 1244) I think that this is a quite invalid assumption about goods, and its characteristics, in the world. Later on I will show that Hardin might be mistaken here with his conclusion. But for the moment I will not continue with my critics and go on with the presentation of his ideas. Hardin uses a simple example in his article to illustrate the long-term and large scale effects of a shared use of the commons. In the following I lean very much on Hardin’s text and present his line of arguments.

Suppose that one herdsman of sheep adds to his herd one more animal. The positive component is one additional sheep with can be considered as a value for the herdsman. The negative component is the little increase of the danger to overgrazing the green which is shared by the commons. If we also suppose, that the herdsman is a rational human being, we have to assume that he will add one more

\[39\] I (and probably also Hardin) refer here to the egoistic person in the sense of the Homo Economicus in rational choice theory. I am well aware that some rational and egoistic action or behavior is everything else than egoistic. Not going to school, e.g., might be a rational and egoistic interest of some school students. In the long run this behavior will turn out to be irrational and
animal, and another, and another \((n_1, \ldots, n_n)\) to his herd. Therein is the tragedy of the commons. (cf. Ibid, p. 1244) Based on this theoretical ideas Hardin comes to the conclusion that “Freedom in a commons brings ruin to all.” (Ibid., p. 1244). This argument, of course, applies only to material things which are limited in size and scope (e.g., land, water, oil, natural resources in general). They do not get only scarce by an “overuse” by the commons they also run the risk of die out completely, in cases of species, and exhaustion, in the cases of natural resources.

Hardin then concludes that property rights are needed in order to avoid *fouling our own nest*. “The tragedy of the commons as a food basket is averted by private property, or something formally like it.” Implicitly, Hardin recognizes that the applicability of the “fencing” idea has some weaknesses. Some goods simply do not offer a potential applicability of fences, so to speak. “[T]he air and waters surrounding us cannot readily be fenced, and so the tragedy of the commons as a cesspool must be prevented by different means, by coercive laws or taxing devices ...” (Ibid., 1245) Even though he seems to recognize the difficulties in guiding the action of egoistic individuals, he still sticks to some kind of “fencing” and affirmative action by the state (or any other institution).

Hardin also recognized the historical dimensions for the justification of rights. With regard to pollution, another main example in his article, he argued, “as pollution became denser, the natural chemical and biological recycling processes became overloaded, calling for a redefinition of property rights.” (Ibid., p. 1245)

I do not want to turn into a complete criticism here but it seems to me that Hardin only had material commons in mind when he talked about the term “commons”. Intangible goods like art, music, literature, software – in short, any expression of ideas do simply lack the capacity to decrease by an overuse. The reading of a nice novel, or the enjoyment of a computer-animation does not decrease the value for other people. Here, we have to keep in mind that the novel and the animation is considered as commons. That means the author is given away his intellectual labor for free. So, this kind of immaterial-commons is also distinguished from the material-commons in the sense that the immaterial is most often not natural. National parks or the Seven Seas are natural. This might also be a major difference towards these to kinds of commons.

However, in the late 90th of the last century another kind of tragedy was introduced by M. Heller. It also involved the concept of the commons but it reaches a fairly different conclusion: The Tragedy of the Anti-Commons.

damaging. From this point of view one may conclude that the action in the first place was not rational.
5.1.2 The Tragedy of the Anti-Commons

Hardin’s metaphor is central to debates in economics, law, and science and is a powerful justification for privatizing commons property. (Heller/Eisenberg, 1998, p. 698) However, Michael A. Heller and Rebecca S. Eisenberg detected in the field of biomedical research a quite different impact of privatizing commons. In their article they “identify an unintended and paradoxical consequence of biomedical privatization: A proliferation of intellectual property rights upstream may be stifling life-saving innovations further downstream in the course of research and product development.” (Ibid., p. 698) The origin of such development was settled in the U.S. in 1980. “Congress began to encouraging universities and other institutions to patent discoveries arising from federally supported research and development and to transfer their technology to the private sector.” (Ibid., p. 698) The major effect of the patenting of research discoveries is, nowadays, the major obstacles for downstream development. The terms upstream and downstream has to be understand as followed. An upstream development is basically understood as scientific research which is intended to aim at new discoveries, while ‘downstream’ refers to the development of a future possible product. The major problem in biomedical research now is, that intellectual property rights (here: patents) form an obstacle to the downstream product development, e.g., a medicine.

Heller and Eisenberg conclude that “[a] resource [here: scientific knowledge] is prone to underuse in a ‘tragedy of the anticommons’ when multiple owners each have a right to exclude others from a scarce resource and no one has an effective privilege of use.” (Ibid., p. 698) But it’s not only that “Privatization of upstream biomedical research in the United States may create anticommons property that is less visible than empty storefronts but even more economically and socially costly.” (Ibid., 698). “Recent empirical literature suggests that communities of intellectual property owners who deal with each other on a recurring basis have sometimes developed institutions to reduce transaction costs of bundling multiple licenses.” I think that such an development is highly doubtful when it comes to software development. The open source way and the established proprietary way of engineering software differs a lot. They are not compatible. So, they lack the capacity of transition as suggested by Heller/Eisenberg.
5.1.3 Conclusions from the Tragedy of the Commons and the Anti-Commons

"The law [is] always behind the time." (Hardin, 1968, p. 1245)

As we can see, the concept of private property and its original purpose, to protect the commons from ruin, as Hardin would put it, causes ruin to the commons itself, in the case of intellectual property in biomedical upstream research, as Heller/Eisenberg would probably put it. What on the first glance seems to contradict itself "fits" actually quite well. The reason is that the concept of property is not 1:1 translatable to intellectual results of labor, as I would argue. Intellectual labor has special characteristics. Ideas, e.g., are not limited in scope or space; land, on the other hand, is. I hold that a society is terribly mistaken if it uses the "idea of protection" in respect of any kind of intellectuals with the inherent assumption that the use of an idea might limit the pleasure or possibilities of other individuals. Of course, there are many other reasons and lines of arguments to take up. I will turn to this in the last chapter.

But what we can say so far is if we accept the conclusions from the discussions of the tragedy of the commons and anti-commons we have to strive for a new way of ensuring incentive for inventions in a digitalized world and we have to ensure that the tragedy of the commons does not repeat itself. So, a discussion on the justification of (intellectual) property rights is needed today more than ever. How do we at best set rules which satisfy the claims of laborers of intellectual work and how do we ensure the interests of the commons, or of the public interest? I think that Hardin and Heller et al. gave an impressive inside view of possible consequences of granting no property rights at all and granting restrictive property rights. Another legal writer of the future of the commons was Lawrence Lessig he argued that "The Internet forms an innovation commons. It forms this commons not just through norms, but also through a specific technical infrastructure. The Net of these norms and this architecture is a space where creativity can flourish." (Lessig, 2001, p. 23) The interdependencies and network effects of different technologies do sometimes hinder us to see through the problem and discover the core of the problem immediately.

But it is not only the technology as such which is at stake. I think it is also worth the analyze the internal value of producing the technology. Software has the capacity to lack other material goods to become a technology. Building a care usually requires metal or plastic. Software requires a computer. I think it is sufficient to say that

---

40Here, of course, I have the effects on software development in mind. I do not deny that protection of intellectual work or ideas is necessary when it comes to industries which turn the idea into products. But even here, as will show later, we have a questionable development.
software as a technology can reproduce itself. From the presentation of the open source movement we saw that the engineering process is the most valuable part for the people involved. The 'laws' and 'rules' which are inherently bounded to the way of this particular engineering are worth investigating from another angle. In ethical theory we have the idea of "virtues". Virtuous behavior is also what is expected and appreciated in open source software development. How this is a matter for intellectual property rights will hopefully be more clear after reading the next sub-chapter.

5.2 The Internal Values of Intellectual Work

When it comes to computer engineering and the most valuable result of it (software) I would claim that also virtue of the profession has to be taken into account. Implicitly, again, Richard Stallman as a former MIT computer engineer makes this point in his philosophy about free software. Software, he says, should be free. Thereby he also refers explicitly to the engineering process. The source code of the software (see definition above, Ch. 4) should be available for peer review and peer judgments. The peer review principles (effectiveness, usefulness, competence, security) are, as I would argue, imbedded in every scientific discipline – and computer engineering is applied science.

"Academic advancement is generally based on success in achieving peer-reviewed publication and funding, and typically include peer review of the candidate’s academic career." The social sciences and the humanities only work and produce valuable results because of this principle. Papers and thesis’s have to be published and considered by the scientific community. This is basically the engine for progress. Social scientist own an idea only in so far as they can claim to be the first one who came up with a theoretical construction or even a whole theory. But these theories do not emerge or arise out of nothing. They are based on previous work which had been done in the discipline or, when it comes to interdisciplinary research, in other fields. So the big steps in scientific work are highly dependent on the work which had been done already. Sir Isaac Newton said one time “If I have been able to see further, it was only because I stood on the shoulders of giants.” This line expresses a principle embedded in every scientific discipline. This is exactly what computer science constitutes, too.

Now, after mentioning the embedded aspects of computer engineering as a scientific discipline I would like to describe the threatening effects of scientific work, where

\footnote{http://www.ethics.ucsd.edu/UC/topics/review.htm, accessed 05/11/2004}
intellectual property rights become restrictive as we have seen in the tragedy of the anti-commons. I will illustrate that the internal virtue of an action in itself may be a sufficient argument in itself for the freedom of objects which are needed in order to practice the action.

The internal values of intellectual work are probably best understood by the definition of the nature of the virtues by Alasdair MacIntyre. MacIntyre developed in his book *After Virtue* the idea that some practices have an internal value. Performing these “practices” has a value in itself. MacIntyre defined practice in the following way:

“By a ‘practice’ I am going to mean any coherent and complex form of socially established cooperative human activity through which goods internal to that form of activity are realized in the course of trying to achieve those standards of excellence which appropriate to, and partially definitive of, that form of activity, with the result that human powers to achieve excellence, and human conceptions of the ends and goods involved, are systematically extended.” (MacIntyre, 1984, p. 187)

An example MacIntyre offers is related to the game (American) football. He says that “throwing a football with skill” is not a practice but the “game of football” itself is. “And [so] is the work of the historian, and so are music and painting. [...] sustaining the human communities ... is generally taken to be a practice in the sense in which I have defined it.” (Ibid., p. 187f) MacIntyre imagines in his chapter on the *Nature of Virtues* a child which he wants to teach chess. First, he ‘pays’ this child with candies to play chess. The incentive of this child then is to play, in fact, for candies. The candy is the incentive for the child to take part in the game. Later, MacIntyre says, when the child is older he will discover other values in playing chess. He will understand the *internal values* of playing the game chess. He will play the game with the right attitude and without getting paid.

The incentive for the person to play chess changes. It is not any more and first of all the external value of the game (getting candies), it is more than that the internal value – which is, e.g., to win without cheating or to *strive for excellence* in playing chess, gaining skills like logical thinking, mastering the complexity of possible maneuver etc. This becomes a value to the person. An external incentive, like money, status or “candies” is not necessary.

Another very strong argument MacIntyre derived from his ‘practice’ definition is that the internal value and good of some practices can only be “identified and recognized by experience of participating in the practice ... ” and “[t]hose who lack the relevant experience are incompetent thereby as judges of internal goods.” (Ibid., p. 42

42I am grateful to my supervisor Professor Collste for the advice to study the concept of “practice”, as well as the distinction of internal and external values by MacIntyre.
This offers in fact a serious question if we apply this conclusion to the open source concept of software development. The chapter about the actual practices of open source engineers gave us a little inside view of how software is developed. The incentive for developing software in this way is not (only) money. Raymond and Stallman, who I mentioned earlier, refer explicitly, like MacIntyre points out here, to the internal value of producing good software. Sharing the source good is a necessary condition. In the last chapter of this thesis I hope to be able to prove that this internal value has to be taken into consideration when one discusses the necessary and sufficient protection of rights in software.

The “competence” or the ability to discover the internal value is only when people are actually taken part in this way of software development. But the decision about how to regulate claims in software is in most cases independently fell by politicians or lawyers. So the experts are not heard while but the claims are recognized and judged from an external point of view.

Another crucial for the application to the model of open source software development is MacIntyres explicit distinction between the qualities of external and internal goods:

"External goods that when achieved they are always some individual's property and possession. Moreover characteristically they are such that the more someone has of them, the less there is for other people. This is sometimes necessarily the case, as with power and fame, and sometimes the case by reason of contingent circumstances with money. External goods are therefore characteristically objects of competition in which there must be losers as well as winners. Internal goods are indeed the outcome of competition to excel, but it is characteristic of them that their achievement is a good for the whole community who participate in the practice." (Ibid., p. 190 [emphasized by A.S.] )

Property rights are power. Property rights can be considered as “external goods” according to MacIntyre’s definition. It is also true that a larger amount of property rights one actor possess limits choice is for other actors. But the social circumstances in which the “practice” (here: OSS development) takes place is socially shaped. The technical infrastructure, the education of the people involved, the legal conditions etc. are socially shaped, and not deterministic. The outcome of striving for excellence can be “measured” by the number of external goods (here: property rights in software). On the other hand, the achievement of internal goods is a good for the whole community, as MacIntyre says.

In the case of software engineering we can easily see now the dialectics between external and internal goods. If a society grants to much power in form of restrictive rights to some actors in the game of open source software it might hinder or diminish the performance of the 'practice' for other participants of the practice. The good for
the community is therefore decreased. I think we can recognize this line of argument from the *Voices from the Grass-Roots* from the prior chapter. It is important to notice here that such a line of argumentation applies only when the internal value of the practice is endangered by restrictive external goods. It does not offer any argument for the abolition of institutionalized external goods of the practice as such but it offers arguments for where to draw the line. Fame, status etc. are perfectly possible external goods which does not limit the possibility for other people to perform the practice.

It should also be noted that a restrictive (property) right does not *per se* limit the possibility of engineering open source software but the freedom of performing the 'practice' and striving for excellence might be threatened because of the dialectic mentioned earlier.

> “[N]o practices can survive for any length of time unsustained by institutions. Indeed so intimate is the relationship of practices to institutions – and consequently of the goods external to the goods internal to the practices in question – that institutions and practices characteristically form a single casual order in which acquisitiveness of the institution, in which the cooperative care for common goods of the practice is always vulnerable to the competitiveness of the institution. In this context the essential function of the virtues is clear.” (Ibid., 194)

I would argue that the open source community can build an institution. The intimate relationship of practices and institutions should be obvious since we studied how open source software development works. And here we finally also find a sound explanation why open source software development has a moral impact. It strengthens the virtues of the people involved.

The distinction between practices and institutions is easy to grasp. Open source software development is a practice. The way people work on software constitutes the unique value of the practice. The institution could be the whole community or the single project. It depends very much for what kind of purpose the software is developed. “Institutions are characteristically and necessarily concerned with what I have called external goods.” (Ibid., p. 194)

Now let me make some final remarks about MacIntyres concept of ‘practice’ applied to open source software. The crucial point of doing ‘practices’ is to discover or get aware of the internal goods and thereby shaping the virtuous behavior of a person. It is important to understand that a ‘thing’ has from an inside view much more facets than from the outside. Arguing that soccer players are fools which “hunt” a ball is far beside the point, for example. Everyone who participated in a soccer game knows how important it is to play with each other. To help each other out in some situations during the match. It is important that the players realize the necessity to show at the match time at the field. That means it is not only the
game ("the ball-hunt") it is the whole impact on the moral development of the participants. Learning to know what responsibility, cooperation etc. means and learning to discover and develop the virtues for oneself. There are many activities which can help to advance the virtuous behavior of a person – and one is for sure open source software engineering. Besides, "[t]he exercise of the virtues is itself apt to require a highly determinate attitude to social and political issues; ... ".

One final remark about MacIntyre’s criticism of utilitarianism.

"Utilitarianism cannot accommodate the distinction between goods internal to and goods external to practice. Not only is that distinction marked by none of the classical utilitarians – it cannot be found in Bentham’s writings nor in those of either of the Mills or of Sidgwick – but internal goods and external goods are not commensurable with each other. Hence the notion of summing goods – and a fortiori in the light of what I have said about kinds of pleasure and enjoyment the notion of summing happiness – in terms of one single formula or conception of utility, whether it is Franklin’s or Bentham’s or Mills, makes no sense." (Ibid., p. 198f)

The of the incapacity of utilitarianism to recognize the distinction between internal and external goods will matter later when I try to develop another view on intellectual property rights in software. Utilitarianism faces many problems. One is, for example, what kind of good should be measured? Here it will indeed a lot of we accept the distinction between internal and external goods and use them in an utilitarian weighing. But more about this difficulties will be taken up in the next chapter.

5.3 Software Development and Virtues

Moral heroes are considered as virtuous. For the state it is and was the question how to educate its citizens in such a way that they become virtuous citizens. Johnson (2001) refers in her book to computer engineers as virtuous persons. Thereby she states that computer professional have a special responsibility towards clients, customers and the public. I hold that this enumeration lakes a very important aspect which is crucial in this thesis. The laboring on intellectual products, like software, requires an even more virtuous behavior of the computer engineers in practicing their very profession since this the only way to keep in check the enormous complexity of computer engineering. Virtue in such a field means to be open for critics from peers and willing to accept and work with these critics, like in any other profession. Gerald Weinberg states in his famous and influential “The Psychology of Computer Programming”: Programming is, among other thinks, a kind of writing. One way to learn writing is to write, but in all other forms of writing, one also reads. We
read examples – both good and bad – to facilitate learning.” and later he says, that “[A] common goal often influences group members to learn together, ... a common product goal need not lead to *mutual learning*. On the other hand, team members always have a common goal, regardless of the product the goal of helping each other learn to perform better.” (Weinberg, 1998, p. 4f and 46)

Then Weinberg introduces a psychological concept which is well known in social psychology as cognitive dissonance. I will shortly explain what this concept contains because its results will prove the importance of virtuous behavior in computer engineering. Suppose you pay in an experiment one group of people $1 apiece and another group $20 apiece to defend a position which is in contradiction to their own conviction. Now, you might suppose now that the $20-dollar-group will produce the best results. If you do so, your wrong. The $1-dollar-group produces the best results. Who is this possible? First, it should be said that this “effect” was validated by many different experiments in social psychology. It is possible because the people in the $20-group get paid to develop an argument which contradict their own conviction. People in the $1-dollar-group do not have that incentive, which is money. But since the task is to write something which is against their own opinion and money is not the biggest incentive here they try to get something out of the counter-argument. Why this matters also to a good deal in computer programming had been describes by Weinberg as follows:

“Now, what cognitive dissonance has to do with our programming conflict should be vividly clear. A programmer who truly sees his program as an extension of his own ego is not going to be trying to find all the errors in that program. On the contrary, he is going to be trying to prove that the program is correct – even if this means the oversight of errors which are monstrous to another eye. All programmers are familiar with the symptoms of this dissonance resolution – in others, of course.” (Ibid., p. 55) Weinbergs conclusion then is that “[a] programmer who truly sees his program as an extension of his own ego is not going to be trying to find all the errors in that program.” (p. 55) This is quite interesting to note when it comes to the justification of property rights in software. If we consider for example a Lockean account it is questionable if the need to a property right is really urgent as described in his Second Treaties as it seems here. It seems that not every resource which has been mixed with one’s labor justifies a property right as we understand it nowadays. Of course, computer engineers mix in a way their labor with the resource. But since this is mostly only a marginal piece of work of the whole product and since so many other people contribute their labor to this product it seems a little beside the point to choose the Lockean approach for justifying property rights in results of labor. I
elaborated on this argument a little more in the last chapter; there, I hope to be able to provide a more sophisticated argumentation.

Another argument for the importance and recognition of the profession of computer engineering provides the Swiss sociologist Geser. In respect to professions and the imbedded mechanisms for the insurance of quality of the work etc. he states that “scientific [intellectual] work is supposed to be guided by interpersonal communication networks within professional disciplines as well as by systems of worldwide institutional publication.” (Geser, 1996) When it comes to computer software the elementary “text” is the source code of the program. In proprietary software this source code is closed. That means it is not accessible for the outside world. To follow the professional principles as suggested by Geser, this issue would have to be resolved or peer review (institutional publication) is impossible.

My purpose of this part of the thesis was to exemplify that some objects should, indeed, not be owned, even though they could. The always remaining crucial question seems to be how “much” ownership is necessary and useful. Sometimes, as the MacIntyre approach illustrates, ownership should not be granted because of some internal value which are at risk.

43 http://www.geser.net/cowo.htm, accessed 04/30/2004
6 Another View on Intellectual Property Rights in Software

This chapter is will summarize the moral underpinnings of intellectual property protection which had been presented throughout the thesis. The re-examination of the legal concept of intellectual property rights and the justifications of the moral rights, somehow embedded in these concepts, might be a major task of applied ethics in the upcoming years. Kimpaa demanded that “philosophers should contribute more to this discussion” (2004a, p. 483).

6.1 Summarizing the Arguments

In the previous chapters of this thesis I expounded the problems related to intellectual property rights and its application to the immaterial good, software. That is,

(1) questioning the justification of intellectual property protection in software. So far, the moral justification was derived from material property protection. Analogies had been drawn from protection of material goods and were applied to immaterial goods. The tragedy of the commons and anticommons is a reasonable exemplification. Patents in “ideas” are stifling innovations and do not serve necessarily as an incentive for inventions in software.

(2) Trusted Computing (TC) and Digital Rights Management (DRM) are new technical forms for the protection of intellectual property. They do not have to be enforced by law. TC and DRM are purely technical solutions to protect intellectual property. The moral issue lies in the fact that TC and DRM are the direct result of efforts from proprietary digital media parties. That is why I hold that TC and DRM do not have a moral force per se. They are means to protect not only proprietary media but are also an attempt to reduce the distribution of free digital media. Since
intellectual property rights and its moral justifications are dialectical intertwined, it is insufficient to accept pure technical mechanism like TC and DRM and leaving the moral questions embedded in these tools open (or unanswered).

(3) The concept intellectual property is a misleading term, as I mentioned earlier. It contains particular forms like patent law, copyright law, trade secret laws which were established to protect goods of completely different natures. One difficulty is to define of what kind of nature the good (object) is. This is what I attempted to illustrate in the the first part of the third chapter. One question, which is still unanswered, is if software is patentable; that would presuppose it would have comparable characteristics like a machinery.

(4) A much more difficult question to answer is if society should allow exclusive rights to particular goods (objects) at all—even though the protective instruments are available. Here it is essential to question the consequences of the applicability of intellectual property rights, e.g. patents to software, and not only the technical possibility of such an application.

(5) A more “qualitative” approach was illustrated in the fifth chapter about the internal value of intellectual work. Open source software development is performing applied science. Scientific ideas have to be free in order to be fruitful for the sciences.

(6) A last and powerful argument was present quite recently by the law scholars Lawrence Lessig (The Future of Ideas, 2001) and Yochai Benkler (Coase’s Penguin, or, Linux and the Nature of the Firm, 2001). Benkler, for instance, argued that the internet changes fundamentally the way people cooperate and behave. With respect to free and open source software he argued that

“The emergence of free software, and the phenomenal success of its flagships – the GNU/Linux operating system, the Apache web server, Perl, sendmail, BIND – and many others, should force us to take a second look at the dominant paradigm we hold about productivity. That paradigm is that production is organized in only one of two forms – market-based exchanges or firm-based hierarchies. Both these traditional forms of production depend on clear property rights to control resources and outputs.” (Benkler, 2001, p. 3f)

and Benkler concludes

In this paper I suggest that peer production of information is a phenomenon of much broader economic implications for information production than thinking of free software alone would suggest.” (Ibid., p. 40)

For regulators, the implications are quite significant. In particular, the current heavy focus on strengthening intellectual property rights is exactly the wrong approach to increasing growth through innovation and information production if having a robust peer production sector is important to an economy’s capacity to tap its human capital efficiently. Strong intellectual property rights, in particular rights to control cre-
ative utilization of existing information, harm peer production by raising the
cost of access to existing information resources as input. This limits the
capacity of the hundreds or thousands of potential contributors from
considering what could be done with a given input, and applying them-
selves to it without violating the rights of the owner of the Information
input.” (Ibid., p. 41)

Benkler’s approach is fairly revolutionary but it is sound and I am inclined to agree
with him. If a society is really better off if it does not protect an object by any
means or at the cost of “new industries” than one really has to consider to abolish
intellectual property rights in those objects.

A similar line of argument was presented by Kimppa (2004a): “A Consequentialist
Consideration of Intellectual Property Rights in Software and other Digital Dis-
tributable Media”. I hold that Kimppa is moving with his approach in a direction
which is finally pointing to the real issues of intellectual property rights in soft-
ware. So far, intellectual property rights have been treated and justified by either
liberatarian (Lockean) lines of arguments or utilitarian.

Many authors so far were mostly concerned with the sets of rights an owner of a
software should be entitled to and how powerful a right should be. This approach
has some weaknesses. Nissenbaum (1995), for example, is raising the question in
her article if copying is immoral. This presupposes that the software is protected
by copyright AND that the copyright (see: terms of use, terms of condition) is
forbidding copying. Nissenbaum uses utilitarian arguments, too, and tries to analyze
how restrictive the copyright law should be and what law would cause the best
consequences for the society and would at the same time reward sufficiently the
author of the software. These issues might have caused important moral questions
10 years ago. Besides, one has to keep in mind that a copyright law only frames
the rights of an author and legalizes the enforcement of a copyright. It sets, for
instances, the rules to what kind of “thing” a copyright is applicable. It does not set
the terms of use of a copyrighted in the “thing” like a software program. However,
today, the major question is not anymore if copying of software is immoral. It is
about the question how to ensure the possibility of open source software developers
to produce software which can be copied freely.

Here, Kimppa is suggesting a new – consequentialist – way. The value of his
approach is embedded in the quality of his investigation. As we saw in the fourth
chapter of this thesis software development changed significantly and we simply have
to admit that some assumptions about the future incentive for software development,
which were held to be ensured by intellectual property rights, are mistaken. That is
why Kimppa chose to take qualitative consequentialist considerations into account.
The question is not, as I said, to ensure the incentive and rights of copyright holders
but to make sure that any form of "intellectual property production" can flourish. Kimppa expressed this in the following way:

"I will consider that in consequentialist thinking 'as much good for as many as possible' the 'for as many' seems to have been forgotten. [...] I will point out, that not all innovations are qualitatively equal in importance for the users. Those innovations that stem from the need of the innovator, rather than the need of the marketing department, rise from concerns of what is needed rather than the concerns of what can be marketed. I will also approach the question of whether IPRs [Intellectual Property Rights] in Software and other digitally distributable media indeed grant limited monopolies as claimed by their advocates or actually grant unlimited monopolies at least for all practical purposes, and what are the consequences of that. " (Ibid., p. 484 [emphasize by A.S.])

The blindness of consequentialist approaches to the (Kantian) internal values (MacIntyre) of intellectual work were underestimated, as I would argue. On the other hand, as I would argue too, knowledge an information has to be free in order to be productive. The key to this utility (productivity) is embedded in the freedom of knowledge and information. Kimppa says about utility that it "is not measured only in monetary rewards. [...] if we are to believe the economists, it would seem (to large degree) be the reasoning that only quantitative amounts can be measured. The more innovation we get the better." (Ibid., p. 485)

"I will start by showing that the main utilitarian argument that we should try to maximize the good of the society has been misunderstood to mean only quantitative good measured in [...] profit. Not enough attention has been paid to the qualitative aspects of good, of understanding what constitutes good for the members of society and whether it is equally transferable from one person to another through some medium such as money. (Ibid., p. 484)"

Copyright law and patent law, on the other hand, are getting more and more the basics of business models. That means companies apply for patents of inventions which are "obvious", non-"novel" and might be applicable to a number of other industrial branches. Then these companies earn their money when an infringement of the registered patent occurred. Therein is their business model. In the literature these patents had been labeled "trivial-patents".

The purpose of intellectual property rights in general is to protect a useful invention/innovation. The commercializing of intellectual property rights is one of the most dangerous developments from a moral point of view for society because the risk is quite high that intellectual property monopolies and trusts will emerge. Kimppa points out, that innovations, on the other hand, which are a result of a demand or need by people would have been very probable developed anyway. Herein is also Kimppas concern about the "as many" in "as much good to as many as possible".

Who is the target of good; the citizen or the corporation?
“The future of the industry – both in content and software production seems to be forming on the rules of the content industries and technology industries.” (Kimppa, 2004a, p. 486) As we will see, this is a powerful argument to reconsider the intellectual property rights in software and in digital media in general. “Other stakeholders, namely the scientists, developers and the general public seem to have been forgotten when changes in the copyright protection are made.” (Ibid., p. 487) Kimppa continues then I claimed that the source code of proprietary software can not be studied and security holes cannot be fixed. I argue that it should not matter if this can be studied by an customer (e.g., a government) or not since the customer has the possibility to choose which software he/she/it will use. Free and open source software, for example, allows explicitly the access to the source code. The danger is in restrictive intellectual property laws which would forbid the development in the open source way.

Kimppa mention also another difficulty in respect to restrictive intellectual property rights in software. He says that the:

“Developments and distribution of software (and other immaterial) to become far more expensive than it would be if no IPRs were in place, thus implicitly supporting the need of IPRs because it is easy to point out that development is expansive, difficult and time consuming and thus needs to be rewarded by IPRs – prime example of circular argumentation.” (Ibid., p. 487, [emphasized by A.S.])

Here I do not fully agree with Kimppa. The argumentation of proprietary software developers and companies is indeed circular. This kind of reasoning is not leaving the legal sphere and it is not pointing to the moral underpinnings of software. On the other hand we might ask, why not granting restrictive property rights? Copyright law is heavily used by open source software developers. Richard M. Stallman, THE advocate of free software, argued that ownership in software is wrong; but I can see no other way to protect the software written by the “community” from proprietaryizing and exploitation in a software-world with copyrights. It should be possible to protect software by copyright law. What matters are the “terms of condition” of the license. So, in a “circular argumentation” a restrictive intellectual property rights makes actually sense. But, a restrictive property right which would forbid the free flow of information and program code is bad for society. Patent law is capable of causing such an effect. So, the circular argumentation has carefully to be investigated. The analogy here is closely connected, again, to the common/anticommon problem. Copyright law in software is an enhancing tool for society, for the commons. Patent law is not – if applied to software. It is one of the tragedies of the anticommons. The reason for this is in the nature of patent and copyright law. A copyright allows the protection of en expression of an idea (see chapter 3). A patent
grants the protection of a process, improvement, composition of matter. The crucial distinction is that a patent is capable of monopolizing (oligopolizing) a whole industry since a “process” has to be used in different applications which can be copyrighted. In an oligopoly industrial environment this might (will) cause a stifling of innovation – a tragedy of the anticommons.

Another strong argument against restrictive intellectual property mechanism presented Kimppa at the end of his article. “All copied products are often considered lost potential profits even if the users copying the product would not have purchased the product irrespective of whether they were able to copy it or not.” (Stallman, 199444. In Kimppa, 2004a, p. 490). This argument has indeed a force. Many products in the digitalized world are used because they are available. The internet is the best example. If people would have to pay for every content they access – they probably wont access it. On the other hand, and here is Stallmans argument, the profit of society and the content producer for “open-content” might be greater than “closed-content”.

And here is the strongest argument for re-thinking intellectual property rights in digital media – that is in software, too.

Kimppa (2004a) seems to agree with Richard Stallman45, who claims that software should be free, in the sense of having no copyright owner. I agree with Kimppa and Stallman at this point completely. A copyright is an option for a developer in a legal system where copyright exist. This option is needed because it is an option for other developers as well. This might sound a little odd but the argument has, as we will see, a moral force. Copyright might forbid the access to the source code of the software (proprietary software). This software might contain source code from non-proprietary software. To enforce the valid claims of the non-proprietary the non-proprietary software has to be copyrighted. This sound even more confusing. The term “proprietary” is used in the legal writings for software which does not allow access to the source code of the software, or forbids, if access is granted, modifications and publication.

Non-proprietary software is published under different copyright “terms of use” which explicitly allow access to the source code, allow modifications and free redistribution.

This kind of software is also labeled “copyleft”. It is used to ensure and enforce the freedoms of end-users and developers in the software.

Now, let us suppose that there is no copyright law in software (Stallman’s claim). In an “internetless” world software development would be hard. In an internet-connected “postmodern” and “post-information-scarcity” world, information, knowledge, and creativity is available on the world largest network. It can be produced, used, evaluated and re-evaluated on marginal costs. The network effect produces a value which is morally significant for the world-society and therefore for a serious re-evaluation of intellectual property rights.

6.2 Justification of a New Approach to Intellectual Property Rights in Software

The legal concept of intellectual property rights was founded in societies were the internet did not existed. Network effects for the creation of goods were not possible because of the missing “layer”. Lawrence Lessig illustrated the characteristics of the internet in the following way: “Cyberspace [the Internet, A.S.] has a different architecture [different from the real world]. Its nature is therefore different as well. Digital content can be copied perfectly and practically freely. You can move a great deal of content almost freely and instantly. And you can replicate whatever good there is in one place in many place – almost instantaneously. The barriers of cyberspace in its natural state are radically different from the barriers in real space.” (Lessig, 2001, p. 121)

How Lessig characterized the cyberspace is also 1:1 applicable to what software constitutes. You can copy it and “move” it instantaneously. What should be explicitly added here is that it can be produced and distributed in peer production (open source) on the internet (cyberspace).

The legal concept of intellectual property rights assumes that the right(s) of the author has to be protected. Nissenbaum (1995), Snapper (1995) and many other authors were looking for a justification for a valid claims of the author and claims for the society. The major work for an ethicist was to look for a morally justification and balancing of claims from these two poles; the interest of the society and the interest of the author.

I think it is time to question property rights in intellectual work as such and especially in software. Again, Kimppa wrote that “[t]he IPR laws are getting more and more stringent and at the same time the intellectual commons is pushed to a smaller and smaller area.” (Kimppa, 2004a, p. 492) The effect of intellectual property rights seems to be, indeed, monopolizing knowledge and information. And Spinello added that “[i]ntellectual property, which includes ideas and other creative content, does not need the same level of protection as physical objects, since the
consumption of intellectual objects is nonrivalrous, that is, one person’s consumption
does not lessen the consumption of anyone else.” (2003, p. 3)

Software is a non physical object. Its production takes place even though the
“producers” do and will not enforce their rights in it. The Lockean approach to
intellectual property rights in software is, so to speak, knocked out. A new paradigm
is needed to fill this gap. Like chapter 2 and 4 illustrated it is also quite questionable
if a right to exclusion can serve as an incentive for innovation. It seems the incentive
of innovation is embedded in the culture of software engineering; and so it is in
scientific work.

I am well aware that this is a radical conclusion: no property rights in software
because the society would be better off without them. The major obstacles are
embedded in the contemporary developments in the political sphere and in the legal
system. The Commission of the EU is about to allow patent law to computer
programs. The proprietary software industry argued that this is a necessity because
thousands of jobs are at stake if the directive will not be approved. On the other
hand, one could argue that thousands of jobs will emerge if the directive is not
approved.

Kimppa, Benkler, Lessig, Stallman are heading into a direction which is meant to
bring about an all and all good for society – qualitative as well as quantitative. From
an ethical point of view I would like to argue that the abolition of intellect property
rights in software is morally justified. The difficulties such an proposal faces the
legal practice. This thesis illustrated that the condition for software production
changed. It also seems that the psychological, political conditions changed as well.
Open source software development proved somehow that rights of exclusion are not
an necessary incentive for inventions and innovations.
7 Bibliography


RAYMOND, E.(2001): The Cathedral and the Bazaar, O’Reilly, Sebastopol


