Software Freedom: An Introduction

Software Freedom:

An Introduction

Robert J. Chassell

This document introduces software freedom: why and how such freedom is important.

Edition 1.03, 2005 Oct 27

Table of Contents

Introduction

Often times, our societies are hurt by the newly gained ease with which computer programs can be duplicated and information stored. We face new and different difficulties than in the past. These difficulties come from bad laws, false beliefs, and obsolete institutions.

With software freedom, we can overcome these difficulties (leaving us with the more 'normal' problems of an advanced society).[1]

When I first started this book, I thought I was only going to discuss the implications of "increasing returns to scale", in which the more you make, the easier it is to make more. As a practical matter, oligopolies or monopolies are the consequence of such a technology.

In the past, with steel making and oil refining in the late 19th century, this kind of economic development caused problems with both quality and opportunity. In the present, software oligopolies have exposed us to machines that need frequent reboots, systems that are vulnerable to simple viruses, and to a wide-spread belief that computers often fail for no good reason. (Such flaws are avoided by good software.)

But, to my surprise, in working on this book, in addition to talking about problems of quality and opportunity, I also found myself figuring out how to defend freedom from assault, how to defend ourselves from enemies without and from criminals within.

Bad laws and obsolete institutions weaken us. Their perceived benefits are short term. Their long term effects bring danger. The consequences are startling. For long term survival, we must change ongoing practices and ways of thinking within our societies.

I first became involved with software freedom in 1984. In 1985, I became a founding director and the first corporate treasurer of the Free Software Foundation, Inc. The FSF was designed as the institutional arm of the GNU Project, an effort to create a complete, free software system. I have been involved with software freedom ever since. Over all, I have been involved in aspects of software other than programming. (I wrote a book, *An Introduction to Programming in Emacs Lisp*, but I do not think of programming as the focus of my life.) I edited more than a dozen books and handled the finances and administration of the Free Software Foundation.

It took many years, but the GNU Project succeeded. GNU/Linux (often shortened to 'Linux') not only works, it is readily available and is used by millions of people. For the GNU Project, the most difficult years were the

[1] For example, the degree to which we should attend to the short term or the long term consequences of an action. One situation occurred in the 1980s and 1990s; the consequences are still with us: on one side, the GNU Project chose not to make it easy for a user, without quite realizing it, to execute code. In contrast, Microsoft Corporation chose to make it easy, as a part of its 'email experience'. This choice is enjoyed by virus writers and hated by those who have suffered from computer viruses.

first seven, in the 1980s and the early 1990s. During that time, we created most of the key parts of the system. In the early 1990s, Linus Torvalds, who was not directly involved in the Project, created a kernel, called 'Linux', which while not as good as the kernel being created under FSF auspices, was good enough and worked. More to the point, Linus placed his kernel under the GNU General Public License. This meant it could be used legally. The GNU system with the Linux kernel proved highly attractive, technically, socially, and legally. Within another seven years, GNU/Linux had spread widely.

It is sad that we had to develop free software intentionally. In the early days — during the first generation after 1950 — all software was free or perceived as free. That is to say, programmers felt they had the right to copy, study, modify, and redistribute it. Indeed, in the beginning, you could not copyright a computer program and you could not patent any of its mathematics. Trade secrecy existed but was not felt as onerous.

Perhaps the best metaphor for software is that of a cooking recipe. A recipe tells you how to cook dinner. A computer program tells a computer how to act. Neither are necessarily considered 'literature', although both may be well written. Most importantly, people expect to share cooking recipes. Although a few cooks attempt to keep secrets, most are happy to tell you what they did. Similarly, at the beginning, most programmers were happy to tell you how they programmed. But then, some lost the freedom to talk.

Beginning in the 1970s and early 1980s, it became legal in the United States for companies to copyright computer programs, and legal for them to patent mathematical procedures. Software vendors stopped supplying source code. 'Non-Disclosure Agreements' prevented programmers from helping each other.

These attacks on programmers' freedom inspired Richard Stallman, a brilliant computer programmer, to talk about freedom and then to pull together many people and start the GNU Project, to create and distribute software that would be free.

This book is not about the GNU Project, nor is it about software as such. Rather, it is about the beliefs, laws, and institutions that surround or should surround the software industry.

And freedom, as with every social institution, needs beliefs, laws, and people to protect it.

Acknowledgements

I am especially indebted to the works of Douglass C. North and Yochai Benkler. I would like to thank both men for their writings. Their works led me to change how I think. Neither, as far as I know, have suggested that they hold the opinions I present here; they may well hold different opinions or have come to different conclusions than I.

I am also indebted to the works of Alan Page Fiske, Adam Przeworski, Lawrence Lessig, and Clayton M. Christensen, who have opened my eyes to the nature and structure of our society and to the institutions that configure it. My many thanks. I hope I have learned well.

I would like to express my thanks and indebtedness to two men whom I know personally, who have shaped my thoughts and life, Richard M. Stallman and Eben Moglen. Richard introduced me and others to the idea of software freedom, and started the project which has led to the most significant current instance of free software, the programs of the GNU operating system using a Linux kernel. Eben inspired me to look further into the institutions and laws that can help or hinder our success.

Finally, I would like to thank Stacey Goldstein and Bradley Kuhn for helping me with this book — most importantly, for keeping themselves and others 'off my back' until I understood myself the issues; and then for helping my own discoveries to others.

My mistakes are my own.

Further Efforts

After reading through quickly and briefly, you may wish to go further. My suggestion is that you determine how certain you judge each claim — perhaps some are weak — and you evaluate the proposals that are mentioned.

For judgement, provide evidence for (or against) each claim. The intent is to exercise 'determinative' oratory, as described in Section "Words Only" in *Choice and Constraint*. However, the evidence will mostly, perhaps entirely, be of the 'I hear' or 'I know culturally' nature, rather than of the 'I reason', 'I observe', or 'I experiment' nature.

Informally, the 'I hear' evidence will consist simply of cocking one's head and saying to one self, "right!" or "wrong!" or "somewhat suggestive". More formally the evidence will consist of scholarly references, news stories, or interviews. The 'I know culturally' evidence requires, I think, a convincing, personal story.

It is good practice to specify what you think is the certainty (or uncertainty) for each claim or bit of evidence: whether it be a slight hint, weakly suggestive, suggestive, or highly suggestive, or the contraries.

Thus, there is text that says that software freedom leads to safety, quality, and opportunity. Is this true, and if so, how true? Is the evidence weakly suggestive, suggestive, or highly suggestive, or the contrary? By evidence, I refer to what you know or can find out as well as to what is written.

Supposing the claim is highly suggestive, as I think. Then you can evaluate a political proposal, in this case, favoring software freedom using the GNU General Public License as the legal tool. An alternative proposal, that I do not mention, is to ban copyright altogether. Yet another is to use the GNU General Public License but to shorten the length of copyright.

To evaluate a political proposal, you can use the 'four Ps of politics': protect, preserve, prepare, and provide. (See Section "The Petals of Cooperation" in *Choice and Constraint*.) You can use other criteria, too.

In each proposal, what is currently uncertain and what should be determined? If the underlying factors are sufficiently uncertain, should the proposal be revised to succeed regard of outcome? If so, how?

1 The Goal

Our circumstances are straight-forward. Over the past half century, technology has advanced. We can now copy and distribute large programs easily and cheaply. We could not do this in the past. This change in technology changes our society.

For safety, quality, and opportunity, we need software freedom. By software freedom, I mean our freedom to use, copy, redistribute, study, and modify software.

Not everyone will choose to exercise their freedom, but with such freedom, more people will be able to discover, develop, and apply advances. We will survive and prosper; those who fail will lose. Similarly, with freedom, more people will have the motivation and resources to make software reliable, efficient, and secure. We will enjoy high quality software. And finally, with freedom, more people will learn to understand, change, and make use of this new world that technology has brought. We all will gain opportunity.

In a small, informal society, a few comfortable social conventions can protect software freedom. But in the large, complex society that we inhabit, we need laws and institutions to defend ourselves; otherwise, freedom will lose. Moreover, as a practical matter, in the modern world, conventions will not be widely adopted by businessmen and others unless they are backed by law.

Firstly, software freedom must be defended against enemies who wage war.

Secondly, software must be defended against thieves. Otherwise they will gain a free ride, and eventually ride us down.

Both these necessities impose constraints on how governments and other organizations may use software, and on their collection and use of information. Moreover, these necessities impose constraints on the types of laws and institutional arrangements that may be chosen.

- Governments, businesses, and other organizations must change how they plan and implement information collection and storage, because the more successful they are at collecting and storing information, the more attractive a target they become to crooks and other enemies.
- Governments, businesses, and other organizations must change how they acquire software and subsidize development, because the more successfully they support restrictions on software, the more vulnerable they become.
- Citizens must change their laws and institutions, to favor safety, quality, and opportunity; because old laws, and the institutions that grew out of them, fail under new conditions.

Fundamentally, it does no good to create or continue laws that aid enemies of any sort or that weaken ourselves.

Moreover, to strengthen ourselves, we need to discover, develop, and apply advances. This means that rather than extend the period of copyright more years into the future, copyrights must be shortened. (Copyright cannot be ended altogether, since it provides a way to protect freedom.) Software patents must be abolished.

Finally, rather than ask a government to purchase or subsidize software that puts restrictions on what people may do, our defense requires that governments focus their purchases and operations on software that frees people to learn and advance, and do this in a manner that protects everyone from thieves and enemies..

All these actions are necessary to defend society, to benefit from software, and to gain opportunity.

These actions will be opposed by the short-sighted, the craven, and by those who wish to attack our countries, to hurt us, and to take away our freedom.

2 Why

There are three reasons to favor software freedom:

- safety,
- quality, and,
- opportunity.

2.1 Safety

For many people, the deepest argument for software freedom is simple: without it, we all will lose our freedom. We will be conquered, if not now, within a few generations.

Victory goes not to the brave, nor to the dedicated but, more often than not, to those with the more advanced technology, and the means and will to use it.

Victory does not necessarily mean military victory; it may mean commercial victory.

An "advanced technology" need not be a computer or other technology requiring a complex production technique. A jujitsu throw may be advanced: the goal is to win. Success may come from a throw that uses an attacker's strength against him and against which he is not guarded.

2.2 Quality

Over time, free software tends to become more reliable, efficient, and secure. Free software gains quality. This is because everyone involved has the motivation, and some have the resources, to improve the software.

Machines should not crash unnecessarily, email messages should not waste their recipients' money, computer systems should not be vulnerable to simple viruses. Computer programs should do what you want.

The owners and managers of a company that produces proprietary software have a motive to delay publishing some bug fixes: the shareholders gain more when they can sell more copies of an update or of a second update; and they can sell more when the update fixes bugs.

Clearly, this economic motivation has no effect when the software – proprietary or free — is sold in a competitive, free market; for in that case, customers will seek reliability. Consumers will avoid a company that delays bug-fix publication.

But if a customer finds it difficult to change, if he or she experiences 'lock in', then the customer may decide to suffer current problems rather than change to different software. As with all people who deal with a monopoly, or partial monopoly, the customer makes a trade-off. He or she knows that the seller is taking more resources than the seller could obtain in a fair market, but the extra cost to the customer may not be enough to stop the customer from using the product.

A customers is 'locked in' when he or she must pay a high transition cost to shift to another vendor or to another solution. For example, a person may keep his or her information, or his or her company's information, in a format that is restricted to one vendor. A 'locked in' customer will accept less efficient software than otherwise, because the loss in efficiency costs less than the costs of change. Indeed, since computers, even those with poor software, provide such an increase in efficiency over the previous generations' pen, pencil, typewriter, mimeograph, filing cabinet, and printing house, a user may not appreciate that his or her poor, restricted software is inefficient compared to other readily available software. Company accountants, for example, may not think that the cost to a company of viruses is a cost they are duty bound to stop.

But 'lock in' cannot exist in a competitive, free market, the kind of market that software freedom creates. In such a market, a customer can easily find another vendor. A customer can easily reduce the costs of viruses.

With freedom, a customer will pay little or no extra cost to shift to another software provider, another service, or another hardware provider. This means that customers will tend to choose the more efficient over the less efficient, since the more efficient costs less over all.

When given the choice, customers prefer security. They do not like the cost of losing their work, or losing a part of it. They do not like to clean up from a disaster. They do not want to be robbed. They do not want their personal habits made available to a crook, their medical or financial data taken, their bank account emptied, or their identity stolen. They dislike the consequences and they dislike the fear.

Many customers dislike insecurity; but if the cost of shifting is higher than the benefits they see to staying, then they will not shift. Worse, some customers have not spent the necessary 'indirect costs' of learning their circumstances. They do not realize that they need not suffer. They have not invested the resources needed to learn that security is possible.

But again, when transition costs are low, people will choose software that is safe. Not everyone will so choose; but enough people will choose so that secure software exists and expands. In addition, hardware providers will offer software that, along with necessary hardware and social habits, gives security.

Only when the perceived benefits of insecure software are greater than the cost of changing, will people stay with insecure software.

2.3 Opportunity

A free society is better than the alternative. You gain opportunity. Your freedom to create and use software is a part of living in a better society. Software freedom guarantees you the legal right to work, to start a business, to choose whom to hire, to chose from whom to buy, to choose software, to help others, and to share.

Software freedom does not guarantee that you will do well or be well; but it does guarantee that you will have the opportunity.

Software has become an important part of a modern economy, like steel or wheat. In the 1950s, the word 'software' did not exist, although what we now call software was being written and used.

Incidentally, I first heard the term 'software' in 1966. Although I understood that the term was a metaphor based on the word 'hardware', at first I was continually confused. I kept mistaking the word 'software' with the phrase 'soft goods', which refer to products such as draperies. I had a hard time learning to define 'software' as insubstantial programs rather than as the kind of product from which you can make substantial, but flexible, curtains.

Advances in technology have brought software into view. Just as we no longer live in the kind of world that existed in the 1450s, we no longer live in the kind of world that existed in the 1950s. In the 1950s, for example, mechanical devices controlled the engines of most automobiles. Now the job is done by electronic devices, which are controlled by software.

The issue at hand is how much freedom we all should have in learning to understand, change, and make use of this new world.

If you or someone else wants to use a car with a particular device in it, how expensive should it be? Should the device be priced at a higher than fair market value, which transfers value from you to the seller, or should it be priced at a level that motivates the seller to sell, but which does not transfer away as many of your resources?

If you or someone else want to start a business involving these electronic devices, how expensive should it be? Should you be forbidden? Should others also be forbidden?

Should you have the ability to choose readily the person or entity from whom you buy services or hardware? Or should you make decisions based on a high cost of shifting? By cost, I do not mean only direct costs, but indirect costs, such as those of changing habits or those of relearning. I mean, all the costs that come into a decision.

Of course, you will not be interested in most opportunities: I, for example, am not much interested in the auto-parts industry, although I recognize its importance. The question is whether you and others should have the right both to be interested and to act.

The different choices open to you and others depend on how institutions are organized. For example, a society can implement institutions that restrict what you may learn or do; or the society may implement institutions that encourage you to learn and act.

It goes without saying that people in a society can only implement institutions within certain constraints; if they wish for more utopian institutions, they may not be able to sustain them successfully. Utopias often fail. Human character, habits, environments, the available technologies, costs, and

outsiders, all limit what can be done. Nonetheless, people have choices, especially in the richer, technologically more advanced societies.

The issues with software are whether to set up institutions that restrict who may use a software package, which raise its cost, a practice that is advantageous to some, but not to others; whether to set up situations which reduce copying, which again is advantageous to some, but not to others; and whether to introduce rules that reduce redistribution, study, and modification?

As the Nobel Prize winning economist, Douglass C. North wrote[1]

. . . institutions basically alter the price individuals pay . . .

For example, legal barriers to inexpensive use, enforced by police and courts, lead to higher costs for people who use that software.

When there are legal barriers to study, organizations tend to work and rework what they already possess, rather than employ inputs from outside, even though the outside inputs may contain valuable innovations.

However, a society cannot do without institutions; as North also said,

The major role of institutions in a society is to reduce uncertainty by establishing a stable (but not necessarily efficient) structure to human interaction.

Every society will define in one way or another the bundles of rights and obligations, the transfers of resources from one to another, the costs, and the benefits, that its members enjoy or suffer.

In the United States, for example, it is legal, relatively easy, and not too expensive to obtain a copy of Shakespeare's play, 'Macbeth'. This was not true in Britain or its then colonies for more than a century and a half after Shakespeare's death in 1616. During that time, the British government continued to enforce the copyright restriction that the copyright holders had on Shakespeare's plays. That law was finally repealed in 1774.

But repeal can be reversed. Until recently, many thought that it would become legal and perhaps even easy to reprint books first printed in the United States during the 1928 - 1932 administration of President Hoover. But the U. S. Congress changed that previous right. You may not legally reprint such books, except by going through a process that is costly in time, and sometimes money; and even then, you may not obtain the legal right — a right that until a few years ago, you had every expectation of having in the early years of the 21st century.

Freedom provides for one way of structuring a society. As a general rule, a free society provides more people with dignity and power than a non-free society.

---

[1] *Institutions, Institutional Change, and Economic Performance*,
Douglass C. North, 1990,
Cambridge University Press, pp. 6, 22
ISBN 0-521-39416-3 hardback
ISBN 0-521-39734-0 paperback

It goes without saying that a society with freedom must limit the power that some might gain. This is because those people's '**freedom to**' enables them to hurt others, who wish '**freedom from**'. There is an old saying, "your freedom to swing your fist stops at the end of my nose". The freedom "to swing your fist" is a 'freedom to'; freedom from being hit is a 'freedom from'.

Quite clearly, not everyone can have 'freedom to' do anything. 'Freedom from' is necessary. Otherwise, people will get into each other's way, and hurt each other. Programmers cannot hide their code from others and still gain the benefits of collaboration.

That is why the major license for software freedom, the GNU General Public License, can be understood as a protection, as a 'vaccination'. The License is primarily a 'freedom from' license; it protects a programmer from having his or her work taken without recompense. (The License is also a 'freedom to' license; it permits third parties to collaborate, even if an author or other first party turns against sharing.)

3 What is free software?

When I write about software, I am referring both to the programs that run the computer, that is to say its operating system, and to applications, such as electronic mail, spreadsheets, writing tools, and Web browsers. In addition, I am referring to applications that are embedded in a machine, applications that control a fuel injector, or operate a telephone, or control a washing machine.

Free software is software that you may use, copy, redistribute, study, and modify.

These five freedoms constitute a bundle of rights.

Of course, an end-user, someone who uses but does not program a computer, may wish only to copy and use a particular piece of software. But that software is developed and improved by programmers and others who study, modify, and redistribute software. The whole bundle of rights is necessary.

The freedoms to use, copy, redistribute, study, and modify software are not intrinsic to the technology: there exists software that you are forbidden to use, forbidden to copy, forbidden to redistribute, forbidden to study, and forbidden to modify.

In addition, for free software to be effective socially, a programmer, or more precisely, the legal copyright holder of a program, must adopt an obligation to the community: when a programmer fixes or extends work that others have done, and makes that work public, the copyright holder must pass on the same rights that he or she received, so that others may use those fixes and extensions. The free software community calls this practice 'copyleft'.

The converse of this is that others have the same obligation. If you are a programmer, this duty of theirs ensures your access to improvements and fixes to your own work, as well as to the work of others. It means that everyone, including people who are not programmers, will benefit.

The GNU General Public License is the legal tool that provides the most freedom for software. It is the most protective of all the legal tools. It takes away others' freedom to hurt you and others, and restricts them from becoming bullies.

In essence, the GNU GPL forbids you to forbid; you may do everything else. It also forbids others from stopping you.

Consider, for example, the history of the GNU Compiler Collection, GCC. This program is used by many developers to convert their human readable code to code that runs machines.

Richard M. Stallman began writing GCC in the mid 1980s. At that time, he wrote it to work with just one computer programming language, C, so initially he called it the 'GNU C Compiler'. However, he wrote the beginnings soundly enough that people could adapt it for other languages, which they did. In part, this extension is the result of good technical work; if the pro-

gram failed frequently when people adapted it to other languages, or had the job appeared too hard, no one would have extended the program. But GCC was well designed and implemented, so people with a certain amount of skill could make extensions. But in addition to being technologically possible, extension had to be legally possible, without hassle.

In the late 1980s, programmers created another program, called Kyoto Common Lisp. They did a good job technically. However, that code was encumbered by a restriction that at first appeared minor: if you transferred the code to someone else, you were responsible for registering their name with the creators. This was intended to enable the creators to warn all users of problems and pass on improvements. The goal was good. But the restriction meant that no one would sell CDs with the software on it, because of the hassle of registration. Similarly, few, if any Internet repositories would distribute the code, because they could not be sure that those who copied it would register properly.

So the software stayed stunted. Few used it.

Then the license was changed to the GNU General Public License (a copyright holder may always make such a change, regardless of the previous license) and the software became more widely used. And then it was folded into GCC. Now, it is part of the GNU Compiler Collection and called 'GNU Common Lisp'.

3.1 How is software made free?

Freedom requires a legal and institutional framework. There are different ways to ensure software freedom; the most powerful way is to attach a special copyright license, the GNU General Public License, to the software. This license gives you and others more rights than most licenses.

In essence, the GNU General Public License forbids a copyright holder to forbid. You — a copyright holder, a programmer, or someone who does not care about programming — may do everything else. Your rights and those of others are reciprocal; this encourages collaboration among those who wish to work on the software. The GNU GPL protects programmers from having their work stolen from them and it protects users from being over charged for shoddy work.

Other kinds of free software license exist; some, unfortunately, do little to support software freedom (see Section 12.1 [An Evolutionarily Stable Strategy], page 55). I will talk about these licenses in Section 6.9 [More limited licenses], page 26.

But first, I wish to talk about the way software is created — false dreams and true practices.

4 How to Create Software

An argument against free software is that it removes the incentives needed for program development. It removes the inspiration that programmers need. This argument is based on a common but false theory of human action.

Some think that programmers work only for pay, rather than partially for pay. The theory is based on the false mental model that people work only for money; and that they are not motivated by anything else.

The theory contains some truth. Certainly, for many kinds of economic activity, pay is the most important motivator. No one works on an assembly line for pleasure. Similarly, those who do not work for pay, volunteers, cannot help all the poor and helpless. Those who claim otherwise are wrong. The burden is too heavy for volunteers to provide the food, the hospitalization, and the income needed. Payment is necessary.

Thus, those who focus on pay as a motivator have reason for their mental model. But it does not apply universally. Those who focus on the benefits of volunteering also have reason.

2300 years ago, Aristotle spoke of the need for an aristocracy. Without one, he said, civilization could not occur. Only those who did not work for material motivations would have time to run a government, to create art, and to write plays.

In Aristotle's day, a group that did not create material objects had to take them from others. Because of the lack of technological development, people, rather than machines, did the work. This is why Aristotle favored slavery, "until the shuttle", as he said, could "weave by itself"[1]. Two millennia after Aristotle, inventors created automated textile machines; these machines could do the work that women, slaves, and other people had done before.

People need to eat to live. A minimal income is necessary. And, of course, people need to learn how to do things. Schooling is required. This costs time and money. So, to follow Aristotle, people who create must have some form of income. They need to be educated.

However, and this is the argument of those who favor volunteer work, in addition to working for paid income, which most economists define as the prime motivator, many people also work for internal reasons, for the pleasure

[1] In Benjamin Jowett's 1885 translation of Aristotle's Politics, Section 1.4, http://www.mdx.ac.uk/www/study/xari.htm#1253b23, the full quotation is:

> *... if every instrument could accomplish its own work, obeying or anticipating the will of others, like the statues of Daedalus, or the tripods of Hephaestus, which, says the poet,*
>
> *"of their own accord entered the assembly of the Gods"*;
>
> *if, in like manner, the shuttle would weave and the plectrum touch the lyre without a hand to guide them, chief workmen would not want servants, nor masters slaves.*

of what they are doing, and for external reasons, to improve their reputation or status in the eyes of others, or because they are expected to work.

Quite obviously, the rich have worked for non-monetary gain, or for indirect monetary gain, for centuries. Some, of course, never work, but many do find that some sort of work is their most interesting long term occupation.

More recently, Abraham Maslow spoke of a hierarchy of needs,

- survival,
- security,
- social,
- esteem, and then, once those are assured,
- 'self-actualization'.

Successful software creation depends on people who can satisfy their needs for survival and security, and who then work for internal or external reasons that are not directly monetary.

Modern technology makes this process, called 'commons-based peer-production', more effective than ever before.

4.1 Commons-based Peer-production

In his essay,*Coase's Penguin*,[2] Yochai Benkler described the successful modern practice for "knowledge work" and named it *commons-based peer-production*.

The keys to understanding commons-based peer-production are that:

- Nowadays, we are richer than we were and more people can engage in "knowledge work" than in ancient times. Instead of women weaving by hand, or slaves cultivating a field, machines can do some of the necessary work.
- Computers and communications have lowered the costs of information inputs. With further progress, the costs will continue to come down, unless governments intentionally create inefficiency.
- Because communications are quicker and less expensive than they were, individuals can experience others' feedback sooner rather than later; this means that even the impatient can enjoy social motivations.

Moreover, commons-based peer-production solves management problems that otherwise dog forms of production based on markets or contracts.

In particular, through peer review, commons-based peer-production provides for 'subsidiarity'. The process locates a decision as close as possible to where it is appropriate. Peer review stays within the commons-based peer-production organization.

---

[2] *Coase's Penguin*, Yochai Benkler, http://www.benkler.org/CoasesPenguin.PDF

From past experience with academics, novelists, artists, and aristocrats, we can understand the key organizational factors for successful commons-based peer-production:

- Provide enough pay. People must survive.
- Reduce generalized fear, such as fear of losing one's income.
- Reduce the particular fear that thieves will steal one's work, possibly in a manner that is felt to be unjust even if legal. (This tells us the importance of the GNU General Public License.)
- Provide for inexpensive communications.
- Provide for appropriate tools.
- Reduce the cost of inputs (and remember to include hassle as a cost).

In addition, we know that:

- People must choose themselves to work on a project, and be self-directed; otherwise, for this kind of "knowledge work", they will be less productive.
- Before going public, some person or group must take responsibility for initial design, to give the project a distinct form and direction.
- The design itself must possess strong and stable foundations, so others can add rooms and towers, without the edifice falling down.
- The design itself must be modular. Modules minimize the cooperation needed for small projects and enables many different and independent people to cooperate on larger programs. Modules enable people to handle complexity.

A "time and motion" expert can tell a person how most efficiently to assemble a widget, or to dig a ditch.

Indeed, a century ago, Frederick W. Taylor, an early "time and motion" expert, became famous for figuring out how to improve the efficiency of digging ditches by hand. Shovels existed in Aristotle's time; and people dug ditches. But a shovel's design, and the way people dug, were inefficient. More than 2000 years after Aristotle, Taylor figured out a more efficient way; the method was adopted by the Bethlehem Steel corporation and then by others.[3]

But programming is not ditch digging. Programming requires more thought and creativity. "Time and motion" are not enough.

Moreover, people who work on software projects vary greatly from one another. In a group of 11 programmers, one may do as much work as the other 10. Also, we know that it is difficult for managers to determine who is able to do more programming than another and the relative quality and

[3] Taylor did two things: after discovering the most efficient weight (from an employer's point of view) for a shovel load, about 10 kg, he designed and manufactured different sized shovels for different density substances; and he devised a way to leverage a shovel off one's hip, so diggers consumed fewer calories.

difficulty of that programming. We know that it is difficult to determine who is motivated at any given time.

Because people differ so much from each other, project managers have a hard time deciding how to pay a fair amount to a person who programs part of a project. This is the case whether that person is in a corporate work force or working through a market.

A decade ago, Richard Greenblatt, a famous programmer and the founder of Lisp Machine Inc., tried to set up a just, proportional-pay scheme:[4] his goal was to design an institution in which many people could cooperate on a project, and in which each would be paid for his contribution. However, no one could figure out how to pay each person justly. The problem continues.

One suggestion is to pay a person according the number of lines of code he or she writes: the problem is that cheats will write more lines; and, in any event, the best code is often well thought out, and short.

Another suggestion is to use contracts: that is to say, to set up a company that hires programmers and that also sets up a management structure to decide and inspire the programmers to work. This is a common technique and is the method that many people assume is the norm for the industry.

Unfortunately, the contract/contractor method is inefficient. Some software projects fail spectacularly. Others cost more than expected.

Moreover, the relationship between customer and vendor may be warped: often, with software, a customer learns what he or she wants over time. Either the customer pays more than planned, or the vendor absorbs the extra cost. Neither are efficient.

4.2 The Advantages of Commons-based Peer-production

As Benkler says in his article,[5]

> *[Commons-based peer-production] is better . . . at identifying and assigning human capital to information and cultural production processes. In this regard, peer-production has an advantage in what I call "information opportunity cost." That is, it loses less information about who the best person for a given job might be . . .*

A person chooses where to work, which increases his or her motivation.

And others encourage that person to work where the programmer will be most helpful. Or, to be less diplomatic, others discourage the incompetent. (All successful projects must include ways to weed out low quality contributions.)

Without the costs and hassle of defining and enforcing property and contract rights, the costs of organization go down. To use economists' jargon, transaction costs are lower.

---

[4] Personal communication

[5] http://www.benkler.org/CoasesPenguin.PDF

Because the goal is to satisfy oneself or others, education benefits. Instead of a customer who pays more than planned, or a vendor who must absorb unexpected costs, people can learn from each other, and help each other, which people like to do when it does not cost them. (Of course, not everyone wants or likes to learn or help others; but enough do.)

Moreover, as Benkler says,[6]

> *Peer-production has an advantage . . . because it allows larger groups of individuals to scour larger groups of resources in search of materials, projects, collaborations, and combinations than do firms or individuals who function in markets. This is because property and contract impose transaction costs to limit the access of people to each other, to resources and to projects . . .*

The art of government is to design the rules and institutions of an economy so that they work well. The rules and institutions must support programmers and others.

In particular, while some programmers do not mind when others take their work from them and do not provide any help or recompense in return, other programmers, the majority, interpret such actions as theft, whether or not it is legal. Theft discourages these programmers. They want 'freedom from' theft.

Hence, a government must provide courts and inexpensive mechanisms to enforce an anti-theft license, such as the GNU General Public License. Otherwise, a government will see fewer programs developed in its country.

In addition, a government must keep down the costs of information inputs to programmers. Otherwise, as in any occupation, higher costs will reduce output.

Finally, a government must never hinder cooperation, else it raises transaction costs and destroys the efficiency of its software industry.

These three requirements tell a statesman what is needed:

- First, a government must ensure that people feel they are living in a society with justice. People work less hard and less creatively when they expect they will be robbed. This is human nature.

  People may work because they have to make a living, but if they feel 'ripped off', they do not do a good job. After all, the virtue of a slave is to rob his owner and to rebel against an unjust and forced labor. (A slave can do this most safely by acting stupid and forgetful, hence slaves' infamous reputation among their masters.) On the other hand, the virtue of a citizen is to reduce the profits of crime, to avoid helping a thief.

- Second, everyone must have the right to use software and to study it without fear. Of course, not everyone will study it; but the question is whether people dare study.

[6] Also in *Coase's Penguin*.

If a person may look at code, but fears that by doing so, he or she will be sued in later life for taking an idea from it, or fears that his or her company will be sued, then he or she will not study. Or if the programmer does study such code, he or she will find — as happened in the U. S. in the early 1990s — that some potential employers will not offer a job, because they fear being sued if they hire the applicant.

Incidentally, the right to study means the code must be available in a humanly readable form — otherwise, as a practical matter, it is no good.

And finally, since the goal from a government's point of view is to inspire more and better software, a statesman will realize that

- The government must encourage modification and redistribution.

  Unless forced, programmers will not intentionally make software worse. On the contrary, they will try to make software better. Similarly, no end-user will select a worse program. If proposed changes look good, chances are that others will accept them, but they will do this only if they are permitted to receive them. If modifications are banned, or their redistribution is banned, everyone loses.

  This means that both by law, and by its own purchasing and funding decisions, a government must favor free software. Otherwise, it defeats itself.

Put another way, to become more successful, a society must reduce its costs, either of transformation or of transaction or of both.

Douglass North looks at economics and economic history among many countries, over centuries. As he points out[7]

> *The total costs of production consist of the resource inputs of land, labor, and capital involved both in transformation of the physical attributes of a good (size, weight, color, location, chemical composition, and so forth) and in transacting — defining, protecting and enforcing the property rights to goods (the right to use, the right to derive income from the use of, the right to exclude, and the right to exchange). . . . The costliness of information is the key to the costs of transacting . . .*

In short, to reduce the cost of transactions, the institutions of society must lower the cost of information. Otherwise, the society generates inefficiencies.

---

[7] *Institutions, Institutional Change, and Economic Performance*,
Douglass C. North, 1990,
Cambridge University Press, pp. 28 and 27
ISBN 0-521-39416-3 hardback
ISBN 0-521-39734-0 paperback

5 Legal framework

The wise citizen converts the general notions of the previous chapter into specific laws, licenses, and other procedures that people can follow.

The necessary specific legal rights are those to use, copy, redistribute, study, and modify software. These rights generate freedom.

It is important to ensure all five of these rights. While you and other people will benefit a little if you have two or three of these rights, rather than none of them, you and everyone else should have all five rights. Without them, you lose the social and technical benefits — you lose them even if you are not a programmer.

It is also important to ensure that copyright holders fulfill their duty to permit others' access to distributed modifications.

5.1 Partial Benefits

In the short run, even a programmer will benefit a little from restricted, proprietary software: the software can run a computer; that is why people use it. But in the long run, a programmer is hurt because he or she is held back. And others are hurt because programmers are forced to fail.

Proprietary software creates a restricted zone; no one outside that zone can learn from such software. No one can improve it. A programmer can learn only that the program solves certain problems; and that it possesses a specific user interface; he or she cannot learn more. Restricted software makes a programmer a dependent who cannot learn and cannot advance.

Worse, such software creates dependency in everyone else, too. Non-programmers cannot use fixes and improvements that a programmer is forbidden to make.

Note that it is fairly straightforward to copy and distribute software that is hard to study and modify. This is often done with products that are restricted, even though such actions are banned. In countries such as Brazil and Malaysia, for example, you can purchase CDs with Microsoft Office on them for far less than the Microsoft corporation charges. However, the current governments are trying to reduce the use of such software, which is illegal. (Free software can, of course, be obtained and used legally by everyone.)

A programmer needs to be able to do more than copy and redistribute software.

For success, a programmer needs to be able to study and modify software. If you are not a programmer, you need to have the legal right to hire people to do this, and to use the software they develop. Or, simply, you need the legal right to choose the software you want, which may be studied and modified by someone else.

The only way to ensure success is a proper legal and institutional framework that protects rights, yours and those of everyone else.

Freedom for software is not a technical or business issue: what makes software free rather than imprisoned is the legal and institutional framework in which people work.

5.2 Need for a Reliable, Quick, and Honest Legal System

Regardless whether you are an 'end user', a developer, a businessman, a teacher, or a civil servant, you must deal with people who are strangers, people with whom you have no connections. This means that you must, if necessary, be able to resort to law to settle a dispute between you and a stranger. Usually, you can avoid such extremes, but not always. Most people are honest and moral, but not everyone.

You need a legal and institutional framework to protect and preserve your rights. If you cannot protect your freedom, people will take it from you.

Without law, there are no practical sanctions. Without law, you must depend on your family, clan, friends, or a criminal gang. For small groups, such help succeeds, but not when you deal over great distances with strangers.

Moreover, the agents of the law must be reliable, quick, and honest. If the police or the courts are unjust, slow, or corrupt, people, businesses, and governmental organizations will avoid them. People will 'hunker down'; they will do less than they might, because action is too dangerous.

Please understand: I do not myself like courts or lawyers. What I am trying to say is that they are a necessary last resort. Whenever possible, most people avoid courts and lawyers.

For the law to work successfully, we need a reliable, quick, and honest legal system. No other way of settling disputes provides justice.

What do I mean by a reliable, quick, and honest legal system? I mean an *independent* and *un-self-interested* court system.

- A legal system that is not the agent of the powerful.
- Nor the agent of the outlaws.
- But a legal system that is driven by
  - a respect for the process of being fair,
  - rather than whether the people who work in it are your friends or enemies.

6 Freedom and Duty, in detail

How can you protect your freedom, and others' duty towards you? The answer is to employ a government. The job of the government is to protect you from those who would try to injure and take from you, and to constrain you from hurting others.

Governments work through laws, which are public statements that tell everyone what the government's police and courts permit and what they do not permit. Governments also work through bounties. Bounties pay for some actions and do not pay for others. Bounties may be direct, but often they are implemented as differential tax benefits or as government purchases.

Laws enable certain practices, and try to disable others. Bounties encourage certain practices.

The GNU General Public License is a way to employ copyright law to enable certain practices, and to disable others. The GNU GPL is a specially drafted copyright license. It is a legal tool. It is a license for a piece of software, for code.

In the following discussion, I speak to "you" as if you are a programmer and copyright holder both. Of course, you may be one or the other or neither. You may be an "end user", someone who uses software that others write. Regardless of who you are, the software industry needs this legal tool, this license, since people both outside and within the industry benefit from it.

The GNU General Public License gives you rights to use, copy, redistribute, study, and modify software. It forbids you to forbid. It also forbids others from preventing you from acting.

In addition, the GNU General Public License imposes on you an obligation, a duty that others expect you to follow, one that a government may enforce through its police and courts. Your duty requires you, after you have published a modification to someone else's code, to give to others the same rights that you received when you accepted that code, so that others may use those fixes and extensions.

The GNU GPL was invented by the GNU Project to protect and preserve free software. (Richard Stallman started the GNU Project in 1984. It led to GNU/Linux.)

6.1 Copyright, Copyleft

An aside: since the GNU General Public License gives you **more** rights than the usual copyright license, it is sometimes called a 'copyleft'.

This neologism depends on the multiple meanings of the word 'right'. The word 'copyright' uses the word 'right' but,

- a 'copyright' license is actually a 'lack-the-right-to-copy' license.

On the other hand,

- a ‘copyleft’ is a ‘right-to-copy’ license.

Before discussing your duty, let me first go through the list of rights that come with free software: your rights to use, copy, redistribute, study, and modify the software.

6.2 Use

First, the right to use software.

Even if you possess a computer, and have software that runs on it, you may be forbidden to do so. Sometimes people say, “In that case, I will run the software without permission.” There are two problems with this.

The first is that as a country becomes more successful, the methods of policing improve.

Schools and universities will check students’ software. Companies will not run forbidden software because their managers fear that a disgruntled employee will tell the copyright owner. They fear that their company will have to pay a penalty. Indeed, the company that supplies me with electricity hired a ‘license compliance manager’ to make sure that the company did not run software in a forbidden manner.

The second problem is that forbidden software is often available only in a binary format. Binary software provides you only with the ability to run a computer. You cannot study, learn, or modify the program. You become dependent.

6.3 Copy

The right to copy.

Not many people own a factory that enables them to copy a car. Indeed, to copy a car is so difficult that we use a different word, we speak of ‘manufacturing’ a car. And there are not many car manufacturers in the world.

Ask yourself, do you own a car factory?

But everyone who possesses a computer owns a software factory, a device for manufacturing software, that is to say, for making new copies. Because copying software is so easy, we do not use the word ‘manufacturing’; we usually do not even think of it as a kind of manufacturing, but it is.

Ask yourself, do you possess a computer?

If you do, you can readily manufacture an entity that is as complex as a car. The entity will be a software package, not a material object; that is because of the way our technology has advanced over the past few centuries. It is now relatively cheap to manufacture complex informational objects, but still expensive to manufacture complex material objects. (Perhaps the course of technology could have been different; perhaps not. I do not know.

All I know is that over the past half century, the cost of manufacturing a complex informational object has dropped dramatically.)

In effect, everyone who owns a computer has become the owner of a factory. In the 19th century, such ownership was rare. Now it is commonplace.

In the political language of the 19th century, the right to copy software is the right to use your property, your own means of production.

6.4 Redistribute

The third of these legal rights is the right to redistribute. This right enables you or others to start a business, to choose with whom you do business, to help a friend, to share.

Without the right to redistribute, the market for software will be neither competitive nor free. Without it, the price for software may be inefficiently high. And without it, governments, police, and courts must either be despised, for failing to enforce laws against redistribution, or must spend taxpayers' money to enforce those laws.

Without the right to redistribute, a country will gain less.

The right to redistribute means that you, who own a computer, a software factory, have the right to make copies of a program and provide them to others. You can charge for these copies, or give them away. Others may do the same.

Redistributed code must include source code. Redistributed binary code lets people run a computer, but prevents you or others from doing anything else. Remember, binary code traps you in dependence. As a practical matter, you and others can learn from source code, but not from binary code.

6.5 Study

Next, the right to study. This right enables programmers to learn.

This right is of little direct interest to people who are not programmers. It is like the right of a doctor to study medicine or a lawyer to read legal text books. Unless you are in the profession, you probably wish to avoid such study.

The right to study means that people in places like Mexico, or Germany, or Thailand, can study the same code as people in Japan or the United States. It means that these people are not prevented from learning how others succeeded.

Bear in mind that many programmers work under restrictions that forbid them from seeing others' code. All they see are the programs of their immediate colleagues and the toy programs of school text books. They are forbidden from studying the programs of their more distant colleagues, those who work in different organizations.

Nearly a thousand years ago, Bernard of Chartres said that the best way to see ahead and to advance is to sit on the shoulders of a giant.[1] But programmers who are unable to see others' code do not sit on the shoulders of anyone; they are thrown into the mud. The right to study is the right to look ahead, the right to advance.

Moreover, the right to study means that the software itself must be made available in a manner that humans can read.

6.6 Source code is vital

Software often comes in two forms, one readable only by computers and the other readable only by people. The form that a computer can read is what the computer runs. This form is called a binary or executable. The form that a human can read is called source code. It is what a human programmer creates, and is translated by another computer program into the binary or executable form.

Actually, a programmer can read binary, but with difficulty; it is seldom worth the effort. And some programming languages use the same text for both the humanly readable and the executable code. Either way, readily readable code is best for humans. People will study readable code.

6.7 Modify

The right to modify is the right to fix a problem or enhance a program. This right enables software to grow better.

For most people, this means your right or your organization's right to hire someone to do the job for you, in much the same way you hire an auto mechanic to fix a car or truck or hire a carpenter to work in your home. If you are a programmer, this means your right to do the work yourself, if you wish. If you are not a programmer, and do not wish to hire anyone, then this means your right to choose the program that best does what you want.

Modification is helpful. Application developers cannot think of all the ways others will use their software. Developers cannot foresee the new burdens that will be put on their code. They cannot anticipate all the local conditions, whether someone in Kenya will use a program first written in Finland.

As Douglass C. North said, in a more general statement,[2]

---

[1] And in 1675, Sir Isaac Newton famously said the same: "If I have seen further it is by standing on the shoulders of giants."

[2] *Institutions, Institutional Change, and Economic Performance*,
Douglass C. North, 1990,
Cambridge University Press, p. 81
ISBN 0-521-39416-3 hardback
ISBN 0-521-39734-0 paperback

> *In a world of uncertainty, no one knows the correct answer to the problems we confront and no one therefore can, in effect, maximize profits. The society that permits the maximum generation of trials will be most likely to solve problems through time . . .*

Put another way, the society that permits more modifications will be more likely to survive over time.

6.8 The Duty to Distribute Derived Source

Under the GNU General Public License, when you create and distribute new code that fixes or extends older code, then you gain a duty: if you redistribute, you must redistribute the sources for your new code under the same terms as the older code. This means that people who use your new version of the older code retain the same rights and freedoms that they had when they used the older code.

No one loses the benefits of collaboration.

It means that if you fix my code, and you publish that fix, I have the right to use your fix. It also means that if I fix your code, and publish the fix, you have the right to use my fix.

6.9 More limited licenses

Most computer programs have licenses. But some licenses do not impose an obligation to redistribute fixes or extensions.

These other licenses, such as a modified or unmodified BSD license, permit a person or company to take software that is itself free, and for them to fix a bug or make an improvement, and then restrict who may use that fix or improvement.

The United States government created the original BSD license. In effect, it became a way to subsidize partially monopolistic companies, since each received code that was paid for by the United States taxpayer. (The code was written at the University of California, Berkeley, under a program mostly funded by the U. S. military.)

The original Netscape Public License was like this as well. You could look at their original source code, but if you contributed modifications or improvements, AOL (now Time Warner), the company that purchased Netscape, had the legal right to take your work and prevent you from using any fixes to it or improvements to it that they made. They could legally prevent you from using software with your own code in it!

While a good many people went along with this license, and they called it ‘free software’, many others refused to cooperate with Netscape. I myself think that this is one reason the new Netscape browser was so delayed: Netscape lost the cooperation of the people they needed at the beginning, the people who are the best in the world, who refused to help them.

I mention all this because the obligation is as important as the rights. For success, a company must contribute to the community as well as take from it.

And while good will is important, and community concern is important, it is only through the law can we ensure that everyone acts upon their duty.

7 Software Dangers

There are three parts to computer security: the system's design, the way people use it, and the *old fivesome*: burglary, bribery, blackmail, bamboozlement, and belief.

The technical parts of design are understood and implemented. With GNU Privacy Guard, for example, you can send email that cannot be decrypted.

However, crooks and spies do not attempt the impossible. Technical success means only that they change their focus to studying how people use their machines and to burglary, bribery, blackmail, or bamboozlement, and to finding people whose beliefs they can employ.

For example, to overcome inefficiencies caused by security measures, people sometimes tell colleagues their pass phrases. This is a workaround. The alternative is to instigate a redesign to make the software both more secure and more efficient. But people do not do this unless the action is simple. They tend to do what is simplest in the short run, which is to be insecure.

Worse, often times, neither they nor their managers look upon the workaround as a symptom of a poorly designed, insecure work place. Crooks and spies, of course, seize the opportunity. They listen in elevators and look through windows. They find pass phrases and enter the local network.

Another technique is bamboozlement. Bamboozlement is a suckers' game. In the past, for example, some businesses provided convenient 'help desks' for their employees. If you worked for such a business, and forgot your password, you could telephone the help desk, tell the service agent your name, employee ID, and date of hire, and the service agent would reset your password and tell it to you. You could return to work immediately.

This convenient help method presumed that no criminal or spy will go to the trouble of discovering an employees' name, ID, and date of hire, and then impersonating that employee. To some degree, the presumption is valid. Most of the time, the method works. Most people are never impersonated.

Moreover, the method keeps out 'riff-raff'. With such a barrier, a criminal or spy will act only if the potential treasure lures strongly. But being on a network, the potential treasure may be much bigger than you imagine. This is often forgot.

The relevant treasure may not be what an employee has or has immediate access to; for a criminal or spy, the employee may simply be an entry or transport point. For example, someone in St. Petersburg, Russia, obtained the sources for much of Microsoft's major software; that person pretended to be an employee; but the employee was not the target, the sources were the target.

Alternatively, the target treasure might be the records of a meeting about take-over bids. Millions and billions are spent on take-overs; secret knowledge can bring a fortune. For a military spy, the target might be personal

information about a large number of an opponents' citizens. Civilian businesses often collect such information to inform their marketing. But the information can be used in other ways by an enemy military.

Only in sports do competitors play fairly. In war certainly, and in business often enough, competitors are unfair. They do not 'play by the rules'. This means that a country, or business, or other social institution, must design itself to defend against the worst as well as against 'riff-raff'.

As for defense: the first rule of system design is clear, and old fashioned: 'never put all your eggs in one basket'. For example, never permit all your information to be collected in one place, or to be accessible by any one method or group of methods.

The reason is simple: if your and others' information can be accessed in one place, or accessed in a multitude of different places by one method or group of methods, then well-funded, well-educated, sophisticated bad guys will try.

(It goes without saying that with well defended valuables, the programmers never work with the system or systems that hold the data, the data administrators check on each other, and the users can never access more than a few records each day. Only an 'adversary' military would have the reason and could afford the cost — perhaps in the hundred of millions of dollars — required to fund a theft. It would be difficult.

(Inexpensive theft is less and less likely. When computers that contained medical information on more than a million United States soldiers were stolen in 2002, all presumed that thieves had taken them only to sell the hardware. None thought that crooks knew enough to sell the data to an adversay for vastly more than they could sell the stolen machinery. At that time, the presumption was correct, as far as we know.)

People often make decisions in terms of their personal experience, or their friends' experiences; but such experiences tend to be local. You will know, directly or indirectly, about local thieves. Not so many people have experience with crackers hired by a mafia in St. Petersburg, Russia, or in New York, USA. But with the Internet, your information is as close to someone far from you as to someone close by.

Moreover, as the military says, 'a point of trust is a place of insecurity'. A point of trust is where someone, or a group of people, may be burgled, bribed, blackmailed, or bamboozled. Or where someone can hold secret and contrary beliefs. Crooks and spies do not go to places that do not matter (except to confuse followers, or for vacation); they focus on key points. And if they do not want to attempt to overcome effective encryption technologies, they employ the 'old fivesome'; they use old methods on new people.

If you want your country well defended, and if you want to feel secure personally, then you must insist that neither a government nor a private company nor any other organization collect your information in one place, or permit information in several places be accessed by an 'interconnect' method

or group of methods. In Chicago, for example, jewelry salesmen took to informing the police where they were going, for protection. But a crook got access to that information, and used it for robberies. By telling the police where they were going, the jewelers were telling the crook where to rob them.

That is the beginning: follow the principle of 'don't put all your eggs in one basket', where, electronically, 'one basket' may also encompass many different physical repositories, brought together electronically.

One complication is that a machine that sends and receives email must be on the Net; otherwise, its purpose is lost. Consequently, such a machine, and its software, must be engineered so that nothing carried in an email message can cause harm. For example, its software should not respond to viruses. An otherwise convenient 'macro' feature is a fault, not a virtue. You should avoid an email program, such as Microsoft Outlook, that implements such a danger.

Traditionally, governments have thought to increase their and their citizen's security by centralizing information. Police and other agencies then access and use this information. Unfortunately, this is the wrong approach. The more successful a government or private business is at collecting information and providing access to it, the more that access is worth to a crook or spy.

A central information repository — or a distributed one that appears 'central' only because of electronic linking — is like a single, central fortress; once infiltrated, corrupted, or captured, the fortress falls. When a fortress contains information, capture may mean 'copy'; there may be no visible indication that anything is wrong. The legitimate users may carry on happily and blindly.

Everyone one knows that a single physical target is a prime target. No one but a fool builds a central information repository as a single physical fortress. Instead, an information repository is spread out physically over many different locations, and each requires a different method to access. This is a trivial but good habit.

Nonetheless, from the point of view of an electronic attacker, such a repository is still a single valuable target. Security comes only by disconnecting segments of the repository from each other, so there are several repositories, and by disconnecting them from networks.

Instead of one hundred thousand people having access to information on which an enemy might be willing to spend one hundred million dollars (as suggested by some recent U. S. government proposals), only one hundred people should have such access. We know that security will fail when 100,000 people are involved; we also know that sometimes, security will be successful when secrets are limited to one hundred people.

Unfortunately, not only do these kinds of defenses keep out enemies, they hinder friends. This action increases the cost of legitimate access to the

information repositories. Segmentation is expensive. Segmentation raises the cost of information inputs to those who try to help.

This extra cost means that governments, businesses, and other organizations must rethink how they do their work. For example, rather than collect all kinds of information about air travelers — who their documentation claims they are, their previous travel patterns, their friends, associates, and neighbors in the same network, town, or street — a security agency must work differently. It should decide that the kind of information it sought in the past is too expensive in the modern world. The security agency should expect failures, because it could not collect the requisite information, because it could not successfully analyze it in time, or because it was corrupted. Instead, it should act differently. For example, the security agency should push for stronger cockpit doors. These will hinder hijackers in a passenger compartment who try to enter a cockpit. And the agency should strive to inspire passengers to defend themselves if attacked.

Clearly governments can and have kept some secrets well. My point is that such endeavors are expensive. Moreover, they become more expensive as they become more useful to the 'good guys'.

Companies often collect many pieces of personal information, each small and seemingly irrelevant, such as how much bread a buyer last purchased. A business may forget that this information, with appropriate processing, can also be useful to a military or commercial enemy. Moreover, as I said before, since information can be copied readily, such information may be collected and used by a business, and at the same time captured and used by an enemy. A business, or its host government, may not recognize the subversion.

Instead of collecting such information, a company should focus on making better use of the aggregated information that comes from its sales and which is anonymous. This is safer for all.

Of course, the change is difficult. People do not like to disrupt their lives. They do not want to change jobs. But we are endangered by the kinds of policing and the kinds of information gathering that was used in the past. Successful security means that those activities be abandoned.

Moreover, these activities must be banned. In so far as a private company can increase its sales by analyzing and applying personal information that it collects, it will do so. Otherwise, those who follow the practice will over time succeed over those who do not. The practice must be outlawed. Only by jointly accepting their need to defend freedom, will everyone succeed. Otherwise, the defenders will lose to crooks and to free riders.

8 What Free Software Brings

Why is free software a successful technology? It is successful because freedom brings benefits.

To software, freedom brings:

- security,
- reliability, and
- efficiency.

To customers and businesses, freedom brings:

- frugal standards,
- a choice of vendors, and
- lower barriers to starting a new business,

To society, freedom brings

- access,
- sharing, and
- empowerment

8.1 What Freedom Brings to Software

Because of the ways people respond to a world of freedom, it brings: security, reliability, and efficiency to software.

Free Software Brings Security

In the spring of the year 2000, a large number of people who used proprietary software from Microsoft were hurt by a virus called the 'I Love You' virus or 'Love Bug'. The vendor had created a system that is foolishly vulnerable.

You can, of course, make free software equally vulnerable, just as you can open the door to any house or business and invite thieves in. But none of the free software distributions that I know are so vulnerable. This is because people want to avoid harm and are able to insist that their vendors protect them. Of course, the free software producers do not always succeed, but on the whole, they have done well.

On the contrary, a proprietary vendor may not care about security, or may come to care later than makes sense. According to *Info World*,[1] the senior vice president in charge of Microsoft's Windows development team said,

> *We really haven't done everything we could to protect our customers . . . Our products just aren't engineered for security.*

[1] 2002 September 5, http://www.infoworld.com/articles/hn/xml/02/09/05/020905hnmssecure.xml

This ill design may have made sense in the latter 1970s, when Microsoft first acquired the 'Quick and Dirty Operating System' that it turned into DOS and the initial Microsoft Windows environments. After all, at that time, microcomputers were standalone machines, like typewriters; they were not connected to a network. Security was not an issue. However, secure design has been important since the 1980s, when many people began to connect to the Internet, which grew in size from that of a small town to that of a city, and then grew larger than any city on the planet.

Free Software Brings Reliability

I do not have much experience with systems that crash, excepting when hardware fails, or I am testing experimental software, or when my sister's husband is working on the electricity upstairs and turns off *all* the electricity.

Programs are complex entities. They have thousands or millions of components. Because the components themselves are mathematical objects, that is to say, numbers and symbols, the components will not and cannot break, any more than the number 3 can break. But the components can be combined wrongly, or the programmer can insert the wrong components, or leave them out. Such bugs cause havoc.

An advantage of free software is that many people — three, four, ten, sometimes more, sometimes hundreds — look at a piece of code. And as the somewhat awkward saying goes

Many eyes make all bugs shallow.

That is to say, one of the many people looking at the code will notice the problem. And it will get fixed. Everyone wants and is rewarded for good, working code. The user does not want trouble; the programmer does not want a shameful reputation. She wants a good reputation.

(Of course, if you are not programming or not hiring someone to program for you, then no one may fix a bug that bothers you. Freedom enables you to make the effort or pay the price, if you so choose; it does not guarantee that another will do the job for you.)

In contrast, as I wrote earlier, a proprietary company that sells updates will have a financial incentive to leave at least some bugs in its code. This is so its customers will have an incentive to buy an upgrade.

I find it odd that anyone would purchase overpriced, buggy code, but they do. They either do not know about alternatives or they see their actions as less difficult than switching.

Or they may not understand that quality software exists. For example, I sometimes see people reboot their computer for no obvious reason, that is to say, for no obvious reason if the software is sound. For example, I have heard that some people reboot their computer when they upgrade their a Web browser. This makes no sense at all.

As a practical matter, no one who uses a desktop machine and has a reliable electrical power supply should have to reboot more than twice a year,

and perhaps not that often. The main reason to reboot, besides hardware troubles and electric power failures, is to install a change in the boot program itself.

If you are testing an experimental version of an application program, and if it does crash, you should not need to reboot. Moreover, your work should be saved for you automatically, or all but a few hundred characters of it should be saved, even if the electric power dies.

Free Software Brings Efficiency

A notable feature of free software is that many applications run well on older, less capable machines. For example, not long ago I ran a window manager, graphical Web browser, and an image manipulation program on my sister's old '486' machine. These worked fine. Text editors, electronic mail, and spreadsheets require even fewer resources than those which use graphics.

This frugality means that people can use older equipment. It means that people can work effectively with newer, more capable equipment. At the same time, manufacturers are building modern, low-end computers that do as much as the older ones, and are not too expensive.

There is no need to acquire expensive hardware to run your software, unless you are doing the kind of job that you could not undertake a few years ago.

Similarly, frugality means that embedded systems can run a core of free software programs that are little different from, or even exactly the same as, those that run on bigger machines. Rather than diminish the quality of important programs, frugality enables designers to remove less important programs. (Of course, no one can run large programs on small machines, but for many purposes, small machines are large enough.)

8.5 What Free Software Brings to Customers and Businesses

To customers and businesses, free software brings frugality, a choice of vendors, and lower barriers to entry.

Bloat and Frugality

Computer programs can be written so that they are 'bloated' or concise; that is to say, they can be written to require more or less code to do the same job. Often it is simpler or quicker for programmers to write bloated code thoughtlessly than write tight code.

However, no one wants to publish thoughtless work. Programmers seek a good reputation. Hence, software freedom discourages bloat, since free software is published. On the other hand, software restrictions keep software secret, or secret within a small group. Bloat is less discouraged.

Of course, these tendencies, one way or the other, do not mean that all free software is free of bloat, or that all proprietary software is spendthrift. These tendencies describe only a way of leaning.

Over time, however, the original writer and others will find it easier to understand, fix, and extend less bloated code; they will have a harder time with bloated code. The less bloated code will do better for the society. (Code can be too concise as well as too loose. When code is too concise, programmers — even the original author — have a hard time figuring out what it does, which means they have a hard time changing it. A key to good software is to write the appropriate degree of conciseness).

Moreover, over time, end-users will find that they pay less for hardware that runs less bloated code. This helps consumers.

Frugal standards

In addition to the way programmers write code that they know others will study, free software brings with it frugal standards.

Standards are the way in which two programs, or two people, connect. One important standard is the form in which electronic mail is written and sent.

A while back, for example, I received an email message about "Access to Information and Communication for Sustainable Development". The message was sent both to me and to correspondents in poor countries.

This 'Access' message was written in '`.doc`' format. It took up more than four and a half times the resources needed to convey the contents. It could have been sent as a plain text message.

Next time you budget for a project, consider paying four and a half times its cost. Then consider whether you would fund it. Next time you pay at a restaurant, take out four and a half times the money . . .

For me the resource use was not an issue because I do not pay by the minute for telecommunications, as many do. But I know that my correspondents around the world prefer that I take care in my communications that I do not waste their money or that of their supporting institutions.

Customarily, people who use free software write their messages in plain text. Not all do, of course, but it is a social custom, one that saves on resources.

Choice of Vendors

Freedom means that you, as a customer, have a choice among those who would provide you with software and associated services. You are not in a 'take it or leave it' situation. You can choose among your vendors.

Perhaps paradoxically, this choice is good for many vendors. Yes, it is easier for a customer to leave.

But this also means that a customer is not frightened of working with a small business that he or she fears may vanish in five or ten years. The customer can move on without trouble, and as a consequence, finds it less expensive and less risky to stay. This contrasts with comments I have heard, where a customer decides to avoid a business because moving from it would be expensive, and the customer fears that the business will disappear.

Also, if customers can readily leave, employees know that they come to the business because the customers like the solutions the business sells. Employees like this, because it tells them they are doing a good job. Owners sometimes like this, since they, too, want to know they are living morally.

The Legal Right to Start a Business

Freedom means that you, as a businessman, have the legal right to start a business. You are not hindered by overly expensive licenses. You are not forbidden.

A quick digression here: restricted software often means you are forbidden to become a businessman or entrepreneur, you are forbidden to start a business. Miguel de Icaza, who started a major international project in Mexico, could never has started with restricted software. He was forbidden to use that software. Since free software is sold in a competitive market, its price is low.

I said earlier, Douglass C. North pointed out that[2]

> *. . . no one knows the correct answer to the problems we confront . . . The society that permits the maximum generation of trials will be most likely to solve [them] . . .*

Moreover, as Clayton M. Christensen says[3] the

> *. . . processes that are key to the success of established companies are the very processes that reject disruptive technologies . . .*

Some companies run themselves well enough that they can invest in a change that counters their current customers' wants, that reduces their profits, and that may not work anyhow. But not many are so good.

A society that intends to progress must provide for dud organizations' easy death. In the case of business organizations, the country must provide workable bankruptcy and buy-out laws. Other kinds of organization, such

---

[2] *Institutions, Institutional Change, and Economic Performance*,
Douglass C. North, 1990,
Cambridge University Press, Cambridge, UK, p. 81
ISBN 0-521-39416-3 hardback
ISBN 0-521-39734-0 paperback

[3] *The Innovator's Dilemma*,
Clayton M. Christensen, 1997,
Harvard Business School Press
reprinted by HarperCollins, New York, 2000, p. 112
ISBN 0-96-662069-4

as non-profit hospitals, need a structure that enables, if necessary, their own easy 'discontinuance'. Governments need to be able to replace themselves; losers must be willing to leave. (Elections serve this purpose when they permit the defeat and replacement of incumbents.)

At the same time, a society needs new organizations to work with the new opportunities. There is where it become important that businesses and other organizations can start legally and inexpensively. Without legality, an organization cannot readily grow beyond a limited size. Without low cost, those who would begin or promote new organizations will start or invest in fewer of them.

Running a Legal Business Less Expensively

Free software means that software itself, a necessary supporting part of a business or community project, will be both inexpensive and legal.

Think of this from the point of view of a business or community group. The organization can use restricted-distribution, proprietary software, and either pay a lot of money it does not have, or break the law and steal it.

I should mention that if a country is a failure, and expected to continue as a failure, no one is going to try to stop illegal distribution. I know a fellow in Africa who says that Cameroon is like this.

Brazil is different. It was once considered a failure, but now various U. S. companies are thinking it is a success. Hence, they are pressing the U. S. government to persuade the Brazilian government to adopt laws against illegal distribution. At some point, the Brazilian government will have to enforce these laws to the satisfaction of companies like Microsoft, or else face trade sanctions.

Indeed, in successful countries, non-free software is becoming expensive for use as well as hard or impossible to study and modify.

Other aspects being equal, a business or organization with lower costs is more likely to survive and succeed than one with higher costs. The proposition is simple.

However, in the past in general, software costs, even proprietary software costs, have been low compared to the total costs of operating a business or other organization. Consequently, few have paid much attention to such costs. But the costs of lock-in and insecurity are increasing. Likewise, the costs of not being able to fix problems readily are rising, as are the costs of not being able to choose among vendors. Software costs, and its legality, are becoming more and more important.

8.11 What Free Software Brings to Society

Free software bring access, collaboration, and empowerment to society.

Access

Free software is accessible. With freedom, a programmer is not prevented by law, by cost, or by other practical considerations from studying, modifying, and using great software written by others. He can build on their work.

For example, a few years ago, some Icelanders wanted to customize a widely used, but proprietary program. They told the vendor that they would do the job at no cost to the vendor, but of course, they needed access to the source code, so they could change it. The vendor refused. Apparently, the vendor considered Iceland too small a country, and too small a market, to deserve any attention.

So then the Icelanders investigated the KDE free software desktop. They found they had both the legal and the practical right to customize it for Iceland and to redistribute it. And they did. Indeed, in the first demonstation that I saw, the programmer shifted back and forth between Islandic and English interfaces.

Access means that programmers in a small country are not banned from work. Likewise, a businessman is not prevented by law, by cost, or by other practical considerations from using software that his country's programmers customize. Nor is an 'end user' forbidden or prevented from working with it.

Collaboration and Sharing

The right to redistribute, so long as it is defended and upheld, leads to collaboration and sharing.

People share when they are not harmed by doing so. People like to help their neighbors. With free software, you are not hurt if you help someone else — you lose nothing, but your neighbor gains. And since you will not hurt yourself, you have every other reason to help your neighbor.

Most people are kindly. Also, most recognize that when they help their neighbor, their neighbor is likely to return the favor.

Free software, by reducing cost and hassle, and by removing hurt, encourages collaboration and sharing.

How Freedom and Competition Work Together

Before I try to explain how a competitive, free market leads to sharing, let me talk about competitive markets and free markets separately.

In a competitive market, a customer has the freedom to leave one seller and go to another. This freedom is not merely legal, but also practical. The cost of changing vendors is not so high as to discourage the customer from changing.

A competitive market may not be free. A government or other organization may have created barriers to business entry. An entrepreneur may not

have the legal freedom to attempt to start a business. However, there are still enough vendors in the market to make it competitive for customers.

In a free market, both sellers and buyers have the right to try to sell or buy products. Everyone has the legal right to try to make a living by selling and buying. But still, a customer may face monopoly. The reasons is that a free market is not necessarily competitive: if anticipated profits are too low, it may not make business sense for a vendor to enter a market. For example, a second, competitive railroad company may choose not to build a new railway line between two cities. The railway company that owns the existing line may have a natural, but not a legal monopoly on the railway market between those two cities.

In a competitive, free market, neither side of a transaction has the power to take from the other more than his or her due. In a non-competitive, non-free market, one side can impose a higher price on the other, up to the point where it is not worth the other paying.

In a non-competitive, non-free market, one side or another sets prices. The side with more power selects a price that some find is not worth paying. These potential customers lose because they do not take part in the transaction. At the same time, even at a high price, others do pay because they find the benefits are worth their extra loss of resources. The monopolist or oligopolist makes extra income through the higher price even though the volume sold is less.

In a competitive, free market, on the other hand, the price paid is high enough to evoke sales, but not so high as to discourage some of the customers from buying.

How Competition Leads to Sharing

First and foremost, competitive, free markets lead to collaboration and sharing.

This outcome is contrary to many people's expectations. Few expect that in a competitive, free market, every producer will become more collaborative and more sharing. Few realize that there will be no visible or felt competition among competing businessmen.

The more competitive a market, the more cooperation you see. This apparently counter-intuitive implication is both observed and inferred.

Sharing occurs when people are not harmed by doing what they want to do. People like to help their neighbors. Consider a small farmer, one among a million. My friend George is one such. His harvest is so small, that there is nothing he can do to effect the world price. His neighbor is in a similar situation. Consequently, if George helps his neighbor, his neighbor benefits, and George himself loses nothing on the price he receives for his harvest. Since George will not hurt himself, he has every other reason to help his neighbor. Not only is George kindly, he also recognizes that when he helps his neighbor, his neighbor is likely to return the favor.

This is what you see in a competitive free market: cooperation.

Visible competition indicates that the market is not fully free and competitive. Visible competition means that at most you are half free.

This is a key social consequence of software freedom: you are not hurt when you help someone else — you lose nothing, but your neighbor gains. Moreover, most people will help another even at some cost to themselves. Also, most recognize that help is often reciprocated. People share.

Empowering Society

People want the opportunity to learn, to work, and to share. This is empowerment. Societies are made up of groups of people within institutions. Societies also need empowerment.

For success, a society must encourage individuals to study, not forbid them; it must encourage people to start new community organizations, not forbid them; it must encourage people to start new businesses, not forbid them; it must encourage people to choose among vendors, so that poor vendors stop doing business.

Otherwise, the society will fail to solve problems. It will fall back.

9 The Social Costs of Restrictions

Freedom leads to benefits; restrictions have costs. Although I talk about the costs in various places, there are three significant costs that people do not always consider. Two costs have to do with the way we bring up children: the degree to which we teach them to be selfish, and whether we discourage them from studying; the third has to do with the loss of discovery. These costs are the collateral damage of restrictions.

Selfish by Law: Don't Share That Toy!

"Johnny, please share that toy with Alice!" How often have you heard a parent encourage sharing?

Our culture encourages parents to teach children to share in some ways, and to be selfish in other ways. Thus, a child is taught that some kinds of entity, such as a toy, should be shared among playmates; but that to share other kinds of entity, or to share in other circumstances, is theft. For example, a child is not supposed to take a toy home from a store, unless it is paid for; but at home, the child is supposed to share that toy with a playmate. Sharing and selfishness are defined by our culture and by our laws.

Currently, there is an effort underway to extend the realm of selfishness, to make it illegal to teach children to share in ways that are natural for children and which were customary. Instead, a child is taught to be selfish, not to share.

A few years ago, the daughter of a friend showed off a colorful and dramatic program. However, I could not but notice that the program came from another person. Her use and possession of that program was illegal.

With non-public software, sharing is illegal. The programs' distribution is restricted and even the youngest child is supposed to insist that his or her friends or parents or school purchase additional copies.

But the child can readily manufacture instances of the program — the child can manufacture instances so readily that we call it copying. I have said this before, but the advance in technology is critical: a process that had been hard before, to duplicate an entity, is now so simple that a four year old can do it.

A culture can take advantage of this progress, or it can hinder it. To hinder it means to add another realm of selfishness; to take advantage means to continue a realm of sharing.

People laugh at the notion of anyone checking up on a child at home, and enforcing any law that forbids the child from sharing. But what if the child is a publisher?

My sister's husband, Fred, set up a Web site for my young niece. She loves it and has learned a great deal. She writes her own stories. And sometimes she wants to show a picture to her friends.

And sometimes, Fred has to tell her 'no'. Depending on the license, some pictures may not be reproduced. Her sharing is banned.

9.2 "Rah! Rah! Forbidden to Study"

By law, teachers must prevent their students from studying the source code to any non-free software that they use.

Non-free software encourages young students stand, drink beer, and shout:

```
Rah! Rah!
Illegal to study!

Rah! Rah!
Forbidden to study!

Rah! Rah!
Illegal to study!

Rah! Rah!
Forbidden to study!
```

This chant is fun, for a short while. But as a practical matter, studying is important. Students should have the legal right to study, and should be encouraged to do so. However, in schools and universities, the use of non-free software restricts what students may study.

As a practical matter, students are more motivated to study programs they use than programs they do not use. People are interested in their own lives. Non-free software forces schools and universities to focus on distant, less relevant topics. Their teaching suffers.

Moreover, by studying sizeable programs that they use, students can learn how to construct such programs — large programs are more difficult to design and build than small or toy programs.

Clearly, most students do not want to become programmers, just as most do not want to become lawyers. But just as student lawyers are encouraged to study, so should student programmers be encouraged to study.

Likewise, textbooks provide pathways into knowledge, and are so used. But textbooks do not substitute for reality. Professional programmers learn not only from books, but from the great programs of our time.

If you are a parent, or a teacher or administrator in a school, you can spend a great deal of time trying to enforce laws against sharing and studying. Or you can teach students to disobey the law. This is a common, but poor way to educate a society. Or you can encourage students to share software. You can come out against selfishness. With free software, it becomes legal both to abide by the law and to share with others.

Also, a school, college, or university that uses free software, need pay less for licenses and less for policing. This is good for budgets.

Ask yourself, do you know anyone who uses software that he or she is forbidden to study? If so, you know someone who lacks freedom, who lacks opportunity. You know a loser.

When a school forbids study, it betrays its mission.

9.3 Raising the Cost of Discovery

Restrictions on software slow progress overall. In addition, they favor large companies that possess large inventories of patents and copyrights. The restrictions encourage those companies to rework what they have, rather than produce the best of new.

Put simply, patents and copyrights raise the costs of gaining knowledge to those who lack legal access to information. Before undertaking a project, a developer must spend his or her time and resources talking to lawyer to determine what is permitted or forbidden (or his employer must do this). He must find out, for example, what sorts of study might get him or his employer sued in 5, 10, or 15 years. She must avoid some of the programming techniques she learned in school. Consequently, such developers find it harder to discover, understand, and apply knowledge.

On the other hand, those who work for companies that have access to patents and copyrights will find it less expensive to work in a manner so as to make use of those patents and copyrights that their company possesses, since they cost their employer less. Indeed, if they do not so direct their efforts themselves, their managers will insist. Those who possess patents and copyrights will "systematically misallocate human creativity", as Yochai Benkler points out[1].

Benkler goes on to make the critical point that,

> *. . . only firms that rely on direct appropriation — appropriation based on legal rights to exclude — can benefit from an increase in intellectual property. Indirect appropriation strategies [such as selling a service, or selling related hardware] do not benefit . . .*

Worse, everyone loses:

> *. . . all strategies suffer some increase in their input costs, because of an increase in the probability that an input they need in their productive activities will be owned by another firm.*

In short, restrictions on software hinder everyone. But they hinder those who work for a smaller business or who are independent more than they hinder those who work for businesses that owns many patents and copyrights.[2]

---

[1] In *Institutional Economics of Public Domain*,
Yochai Benkler,
http://www.law.nyu.edu/benklery/IP&Organization.pdf

[2] In addition, in long sentences, Benkler says,

Organizations that minimize costs by utilizing intrafirm sources of information suffer the least increase in costs, because access to their owned inventory continues to be at marginal cost, regardless how extensive their power to exclude others from it. Organizations that rely on barter may be forced to engage in more aggressive rights acquisition, because an increase in excludability increases the probability that their utilization of a collaborator's information could provide grounds for a strategic suit. This increases the cost of using barter/sharing systems, but not the appropriability of its outputs.

10 Misleading Metaphors

Shoes and ships are produced and sometimes sold. So is software.

Moreover, when software is sold, it is often sold as if it were like a ship or shoe. The understanding that comes from physical objects, like ships and shoes, is applied metaphorically to software, even though software cannot be worn on your foot or dropped on it.

The characteristics of software are different from those of shoes or ships. It is wrong to try to understand software sales as if software were like hardware.

Unlike ships or shoes,

- software is non-rivalrous,
- software is inexhaustible,
- software is easily manufactured,
- software is potentially nonexcludable.

To compare software to ships and shoes is to construct a metaphor or simile.

Much of the time, there is good reason for using metaphors: they provide 'mental models'; they offer a framework for thinking. Metaphors succeed by furnishing ways to think that are based on other knowledge. Often, writers make metaphors 'for effect'; they hope to induce in their readers a change in emotion or in perspective. But metaphors, often invisibly, are also used for other kinds of thought.

The concept of 'property' is such a thought. By extending the concept of 'property' from a rivalrous good,[1] such as a car or chair, to a non-rivalrous good, such as software, people can easily decide what they think should be done with software: they apply to software the rules they apply to chairs.

We have social conventions against stealing chairs; we support police and courts to help those whose chairs are stolen and to find those who committed the robbery; we pay insurance to replace unrecovered thefts.

The metaphorical application of these conventions and actions to a computer program proceeds by the same reasoning: establish a social convention by teaching children in school not to share a program, even though they are capable of doing so; fund police and courts to help those who want to restrict who can use their software.

But the metaphor breaks down at the concept of insurance: it turns out that there is no need to insure for the theft of a program, since a program does not vanish when someone else copies it.

This metaphor is called "intellectual property" or "IP". It is so commonplace an expression that many no longer think of it as a metaphor.

[1] The word 'rivalrous' means that your consumption *rivals* mine. Only one or the other, not both of us, can enjoy the consumption at the same time. The word is defined more thoroughly in the next section.

10.1 Software is Non-rivalrous

A car, a house, a loaf of bread, a shirt, a ship, a shoe: each is an economic product that can be used by only one entity at a time. If I take your car from you, you cannot use it. If I eat your bread, you cannot also eat it.

Economists call such goods *rivalrous*: if I cannot wear the shoe you are wearing, if I cannot eat the same bread you are eating, your consumption *rivals* mine.

Efficient use of rivalrous and non-rivalrous resources

Many people believe the best way to ensure that rivalrous resources are handled in an economically efficient manner is to provide each resource with an entity that is able to exclude others from it. In law, this right to exclude is called 'ownership'. The entity owned is 'property'.

The entity owning the resource is expected to handle the resource in such as way as to maximize returns to that entity. And if everyone does this, all benefit.

As Adam Smith wrote in *The Wealth of Nations* in 1776,

> *. . . every individual . . . endeavours as much as he can . . . to direct . . . industry so that its produce may be of the greatest value . . . neither intend[ing] to promote the public interest, nor know[ing] how much he is promoting it . . . He intends only his own gain, and he is in this, as in many other cases, led by an invisible hand to promote an end that was no part of his intention . . . By pursuing his own interest he frequently promotes that of society more effectually than when he really intends to promote it . . .*

This quotation tells us of the moral idea behind the belief in competitive, free markets.

A major, usually unstated, presumption among those who support competitive, free markets is that institutional constraints will permit people to attempt to maximize returns on what they own. Another presumption is that enough people will benefit from this organization of society that those who lose because of the organization will not be able to change the society.[2] These presumptions have been more or less true in countries that have been economically successful over the past two centuries, but are not true over all.[3]

---

[2] Engels and other early Marxists did not believe this presumption; they thought that the industrial proletariat would grow in size sufficiently that they could eventually vote the competitive, free market people out of government. They were wrong, but this is one reason they proposed following legal methods, in contrast to Lenin, who did not.

[3] As Hernando de Soto points out, in many countries, people cannot prove they own their house. This means they can only borrow from a lender who knows them and who has the means to ensure that the money will be paid back: a mafia. Such people cannot borrow money from a lender who will use courts to take the house if the money is not paid back. In this circumstance, people have less access to capital than they do

Nonetheless, in spite of the suffering and disasters we have seen, Smith's remarks have justified the use of 'property' for the handling of rivalrous goods.

However, non-rivalrous resources are not rivalrous resources. They are different. However much one may support Adam Smith, his advice regarding rivalrous resources does not apply to non-rivalrous resources. A way to organize society for the economically efficient use of rivalrous resources is not good for the economically efficient use of non-rivalrous resources

For example, two people can share a program. Indeed, when you ask someone whether two people can share a program, he or she says, 'of course, it is practical'.

Unfortunately, when people set up laws, or accede to others setting up laws, the laws often make sense only if such sharing were impractical.

This is because lawmakers and others often mistakenly apply the lessons that apply to rivalrous resources to non-rivalrous resources. They imagine, falsely, that a non-rivalrous resource is similar to a rivalrous resource. They presume that if two of us have a program, only one of us can use it at any given time. They act as if a computer program were like a shirt that only one person can wear at a given moment. Hence, they create laws that restrict software to fewer people even though many can share.

10.2 Software is Inexhaustible

The idea of exhaustibility grows out of the ordinary idea of a possession, which is an object that you can hold in your hand, wear, or walk into. Such a possession is physical; and there cannot be too many such possessions in the world, else there is no room for anything else. Such possessions are, in the jargon of economists, *exhaustible*.

A shoe or a ship are exhaustible possessions. You can run out of either. Moreover, if someone takes your shoe or your ship, you lose it. However, software is not like a shoe or ship: if I take your shoe, you cannot wear it; but if I take your software, you still have it.

Software is inexhaustible, as well as non-rivalrous.[4]

---

where title to property is established and enforced by an honest court and strangers can safely lend money.

This means that they cannot invest in the kinds of economic development that depend on capital.

The Mystery of Capital,
Why Capitalism Triumphs in the West and Fails Everywhere Else,
by Hernando de Soto, 2000,
Basic Books, New York
ISBN 0-465-01614-6

[4] Well, a nit-picker will say that our abilities in the universe are limited, and we can run out of electronic storage space. But when we talk about how the 'Dutch build the Netherlands' by creating polders and pumping out the sea, we are not focusing

10.3 Software is Easily Manufactured

Software is inexhaustible because it is easily manufactured.

Everyone who owns a computer owns a factory for manufacturing new instances of software. Everyone who browses the Web, or reads electronic mail, is a manufacturer.

Indeed, as I have said several times, this kind of manufacturing is so easy that we do not use the word 'manufacturing'. We call it copying. But copying is what happens in manufacturing.

In a science fiction story, it may be possible to copy physical objects. Your friend might say to you, 'I like your watch.' You would respond, 'Oh, let me copy it for you.' That fictional world does not, as yet, exist. Perhaps it will never exist. But nowadays you can copy certain objects that are not physical, such as software packages. You can make duplicates readily and easily.

This is a change in technology. In the past, it was neither easy nor cheap. In the 1950s, for example, you could copy a computer program by hand or by typing it on a typewriter, or by using a very expensive computer. You can still copy a program by hand or by typing it on a typewriter, but few do. The price of computers and ancillary equipment has come down. For most people, most of the time, it is easier and less expensive to use the computer.

Also, both hand copying and typewriting involve copying to paper; yet most software copying is not to paper. Indeed, people who do not study or modify software hardly ever copy a program to paper; they use the program, which means they copy it from a CD or other computer or other source to the machine on which they run it.

10.4 Software is Potentially Nonexcludable

You can sell a shoe or ship only if you can prevent others from stealing from you. To benefit from a sale, you must be able to exclude from the transaction those who do not pay you.

Most of the time, most people do not rob. People are protected by social convention. However, some people do rob and some conditions promote robbery more than others. An individual who cannot call upon strong friends, or police, offers himself as a victim. The turmoil and suffering of a war, for example, promotes robbery. Ideologies and belief systems that encourage honesty among 'us' often permit lies and robbery against 'them'.

In the jargon of economists, the ability to gain the value of a sale is called "direct appropriation" and it depends on the ability to exclude crooks from a transaction. (The ability to gain value from a different activity than the

---

on oceanic limitations, but on Dutch success in expanding their land, something other countries do not do. Similarly, for all practical purposes, if we want more software, either we use more of the electronic storage space we already have, or we build more such space.

one for which you are paid is called "indirect appropriation". For example, some programmers create high quality free software in order to improve their professional reputations. They then convert their good professional reputations into higher pay.)

Individual items, such as individual electronic messages, may be encrypted in such a way that only the intended recipient may read or use the message. So long as you and your recipient can exclude others from fooling you into telling them the encryption key, or breaking into your computer or their computer, the message — or the item of software — can be made exclusive.

However, as a practical matter, this is not possible for software or messages that are distributed to many people. The reason is straightforward: somewhere in the chain of transactions, someone will make the encryption key known. (This is the same problem as with widely used databases. A military organization often restricts who has access to its secrets, but that method fails when the goal is wide access. Moreover, the more valuable the secret, the stronger the attack.)

Perhaps an attacker will obtain the encryption key through old-fashioned burglary, bribery, or blackmail. Or perhaps an attacker will fool a user (this is called 'social engineering' or bamboozlement), or the attacker will find someone whose beliefs cause him or her to copy the secret. Or perhaps the maker of a key will do a poor job and someone else will decrypt it. The latter happened to DVDs, which are designed to contain a kind of message, a movie, that many people watch. An interested teenage child posted on the Internet a method for decrypting DVDs.

The only way to provide excludability to an intrinsically non-excludable product is to police the product's use. Such policing is expensive. At the very least, to gain some degree of restriction, a society must change how people think. The majority of those with some kind of power must come to believe in the legitimacy of the policing. Parents must learn to teach a new kind of selfishness. School administrators must budget and pay for multiple copies of programs that they or their students could readily and inexpensively manufacture for themselves. Businesses must hire 'license compliance managers' to make sure their employees avoid the easy and the inexpensive. The society must discourage study, discovery, and application.

11 Metaphors explain the new in terms of the old

In the previous chapter, I talked about the way the social conventions and institutions governing software, a non-rivalrous good, may be made by thinking with the idea of property for rivalrous goods.

Let me explore this way of understanding in more detail: I will discuss how people think about one aspect of computers and software, the Internet.

In this discussion, I am going to refer to quite obvious metaphors, not to ways of thinking that some never think of as metaphorical.

In discussing technology, we use metaphors to link older and more familiar technologies with a newer and less familiar technology.

11.1 Metaphor: Information Highway

In the United States, in the mid 1990s, the phrase *Information Highway* became the most common metaphor for explaining the Internet.

I do not know if this metaphor was so common in other parts of the world as the United States. If it was not common, it is still worth studying, because everyone deals with Americans whose thoughts grew out of this metaphor, and who are sometimes wrong.

The metaphor of the 'Information Highway' takes people's knowledge of highways and invites them to apply that knowledge to the Internet.

What does this metaphor tell people? First, it tells people that the Internet is outside your home or office. It is not inside.

Partly, this is a useful analog, since you need to gain access to the Internet, through a telephone, cable, radio, or other communications device. Similarly, if you own a house, you need to build a driveway from your house to the road. But the metaphor does not help you if you live in an apartment building right next to a public highway.

Moreover, the metaphor does not tell you that you can bring remote computers into your home or office. It did not warn me that when I was in Europe, in Germany, I could get confused with whether I was using a machine across the Atlantic in the United States, or one a few hundred kilometers away in another part of Europe.

Nor does the metaphor tell you that you can create a secure local network that stretches across nations and oceans. This ability is important for businesses trying to grow and for the civil society.

The metaphor correctly tells you that Internet connections may be slow intrinsically, like a secondary road, or suffer traffic jams during rush hour. However, it misleadingly suggests that the system takes up a great deal of physical 'space' that could be used for other things within a city, such as parks. The metaphor suggests that the space in which information resides is limited in the same way as space within the confines of a city. The 'Internet

as Highway' metaphor does not lead people to think of the space required by information in the same way as the Dutch think of the Netherlands, as a land that is built. The metaphor hides useful features.

11.2 Electronic Shopping Mall

A second metaphor is the *Electronic Shopping Mall*. This tells you that the purpose of the Internet is to provide a place to buy things, and it also tells you that private investors will pay to build it.

The metaphor suggests that the Mall will need governmental regulation and freedom, since you cannot run efficient or large markets without both regulation and freedom. The metaphor also suggests also that there will be great opportunities for theft, corrupted regulators, sweat-heart deals, and cozy arrangements.

11.3 Great Library

A third metaphor is that the Internet is a *Great Library*. You can search and find information. Indeed, I find that people are often more likely to use the Internet as a reference library than they are a real library!

The 'Internet as Library' metaphor tells us that many people can see the same information, just as many patrons can borrow the same book from a physical library. This is important to those who concern themselves with budgets.

Moreover, the metaphor tells you to expect a vast range of queries. While most inquiries will focus on the same small list of topics, others, a huge number of them, will focus on subjects you never considered. This has critical business and political ramifications.

In particular, it tells us that censorship, including the often misunderstood technology of filters, is a mistake.

11.4 Metaphors Tell Us About the Internet

These metaphors, limited and troublesome as they are, tell us about tools that use software.

The metaphor of the 'Information Highway' tells us about roads with potholes and weak bridges. We want our electronic networks to be reliable. Highways attract highwaymen, thieves. We want our electronic communications to be secure. Highways cost money. We want our electronic communications to be efficient and use resources well.

The metaphor of the 'Electronic Shopping Mall' tells us about burglary. After all, merchants get robbed.

The metaphor also tells us about the importance of trust in commercial transactions, that our money be good. It tells us about issues of privacy, and the opportunities for monopoly, and the moral importance of a competitive, free market.

The metaphor of the 'Library' tells us to expect a small set of 'most visited' sites, and a large set of seldom visited sites. It tells us that people will want to learn about the oddest lessons. People want the empowerment that comes from knowing. The metaphor also tells us that private funding may be too limited to generate the full range of social and economic benefits that libraries can bring.

In essence, these metaphors lead us to the lessons that are learned from other technologies. The metaphors tell us what we want.

11.5 More Metaphors: Viral Code and Vaccination

Another set of metaphors has to do with illness and vaccination.

When others hurt me, I try to defend myself. But some tell me that this makes them sick. They tell me that I should permit people to rob me of my work. They tell me that I should never try to defend myself.

They tell me that I should stop using the GNU General Public License, a license that vaccinates me against hurt. Instead, I should adopt a license that permits other people to rob me with impunity. They want me to adopt a license that forbids me from fighting back. They want me to give up my right to benefit from a derivative of my own work, a right I possess under current copyright law.

Of course, the language is a little less feverish than this. Usually, I myself am not called 'infectious'. Rather, the legal defense that I use is called 'infectious'. The license I choose is called 'viral'.

In every day language, words such as 'infect' and 'virus' describe disease. The rhetoric is metaphorical. A legal tool is not a disease organism; but it is popular to think of the law as an illness, so the metaphor has impact.

The people who want to rob me use language that says I make them sick when I stop them from robbing me. They do not want to draw attention to the so-called 'disease' that makes them ill: my health and my rights, and the health and rights of other people. Instead, they choose metaphor to twist people's thinking. They do not want anyone to think that I am a good citizen for stopping crime. They want the metaphor to fool others into thinking that I am a disease agent.

The GNU General Public License protects me. The connotation of 'virus' and 'infect' is that my choice of defense gives an illness to those who want to rob me. I want freedom from their robbery; but they want the power to hurt me. They get sick when they cannot hurt me.

To use another health and illness-related metaphor, the GNU General Public License *vaccinates* me; it protects me from theft.

Note that the theft about which I am talking is entirely legal in some situations: if you license your work under a modified BSD license, or a similar license, then others may legally take your work, make fixes or improvements

to it, and forbid you from using that code. I personally dislike this arrangement, but it exists.

In addition to my personal dislike, there is a social reason to dislike legal theft. This is best understood using game theory.

12 Licenses, Game Theory, and Strategy

Game theory is a way of thinking about conflict and cooperation among people, nations, spiders, and other entities. John von Neumann invented game theory in a book he co-authored with Oskar Morgenstern in 1944, *Theory of Games and Economic Behaviour*

The word 'game' in the name comes from its application to human interactions in which some people gain and others lose. In a competitive society, such as the United States, games of sport are common and game theory applies to games. In addition, game theory applies to other kinds of interaction.

When I speak of game theory, I often use the vague word 'entity' because the 'players' need not be human. Trees, for example, may be players. Trees need sunlight; they interact with other trees around them for access to sunlight.

The presumption underlying game theory is that different entities act on themselves and on each other, and do so in ways that can be described, with results that can be predicted (more or less) and measured (perhaps crudely).

The quality of entities' strategies is determined by the results. You can look at the results in two ways: at the consequences to individuals; and at the consequences to all.

Different strategies determine which individuals succeed. The form of the 'game' as a whole determines how well everyone succeeds or fails.

To determine how well everyone does as a whole, add up the total of losses and gains for everyone. A war, for example, is often seen as a 'negative-sum game'. Even though allies in a war cooperate, and some win a war, the total costs exceed the benefits, so the end result is negative.

(Incidentally, many people do not think of a war as a 'game' and find the term repulsive when applied to war; perhaps the theory should have been named differently, such as 'a theory of conflict and cooperation'.)

Other forms of conflict and cooperation are 'positive-sum games'. For example, economists often point out that when two groups each produce what they can most efficiently, and trade with each other, the total product will be larger than when each acts on its own.[1]

Yet other forms of conflict and cooperation are 'zero-sum games', in which the gain of one side equals the loss of another. Thieves often think this way: they say to themselves, 'either he has it or I have it. There is no way we can

[1] Economists praise 'free trade' which is the name of this positive-sum game, but the transition to it may be difficult. To become more productive, a group must specialize. Within the group, some people will have to change what they do. But some people hate to change and others cannot change to another job that does as well. So they oppose this positive-sum game and cause their country to forgo the benefits that are supposed to accrue to all as a group.

work together to create more for both of us. If I take from him, he loses, but I gain.'

12.1 An Evolutionarily Stable Strategy

An evolutionarily stable strategy is a way of acting in a situation of conflict and cooperation such that the entity acting can survive repeated interactions with others. The idea comes from a combination of evolutionary theory and game theory. At first, the idea was applied to the conflict and cooperation that occurs among animals and plants, but David Rysdam[2] applied the idea to how people make choices on their use of software according to the different licenses possessed by different programs. According to this analysis, among the ways that different people use different programs, the GNU General Public License provides for an 'evolutionarily stable strategy'.

Let me start with a quick example to show how games theory works.

Imagine two foolish young men, George and John, probably American, playing a game called Chicken. In this so-called 'game', George and John race towards each other at high speed, each in his own car. The driver who swerves loses while the person who doesn't swerve wins. If both swerve, no one wins. If neither swerve, the two crash into each other and die. When both die, both lose.

To make the situation more clear, let's put the strategies in a grid and assign numerical values to each outcome.

The values are the payoff to George, based on what both do.

[Payoff to Geo.]	John Swerves	John does Not Swerve
George Swerves	0	-1
George does Not Swerve	1	-1000

If both George and John swerve, neither wins. The value of the result is zero.

If George swerves and John does not, George loses a little.

George does not swerve, but John does. George wins. This is the outcome shown in the lower left of the table.

If neither swerve, they crash into each other; both die.

The idea behind an "evolutionarily stable strategy" is to consider a situation similar the one above, but for repeated encounters. Moreover, the encounters are not simply between two entities, but among many entities.

---

[2] In *Open Source as ESS*,
David Rysdam, 1999,
http://www2.fastdial.net/~drysdam/essays/GPL-as-strategy.html
or http://www.kanga.nu/~claw/docs/GPL-as-strategy/

After many encounters, those with the highest scores win. In evolution, this means that those with the highest scores survive, or that their children and grandchildren survive

In programming, this means that a program, or its derivative, survives over the years.

The evolutionarily stable strategy is stable in the sense that anyone who adopts a different strategy does worse and worse. Generally it is the strategy that eventually is adopted by the largest number.

Now, let me apply this form of analysis to software. What you can do with software — legally, and, in many ways, practically — depends on the license for that software — whether you may use it at a fair market price, whether you may study it, start a business involving it, and so on.

12.2 Software Licenses

For the sake of simplicity, let us consider only three types of license. There are more types, but this simplification helps us understand. Moreover, this is a rather good description of the general features of the three major kinds of license.

- `Restrictive license`
  The source code is not available to other programs.

- `Between license`
  Initially unrestricted, but may be restricted: the source code may be inspected and included in other programs at will, but when included it may be restricted.

- `GNU General Public License`
  Unrestricted through-out time: the source may be inspected by all programmers, and may not be restricted. When included within other projects, the licenses of those other projects must also permit the same freedom. The code must remain unrestricted.

Consider what happens in various situations. For this, we think not of programmers, but of whether people who use the code continue to use it. Metaphorically, we say, 'one program meets another'; what we mean is that 'users choose among different programs depending on how their developers improve the programs.'

- **One restrictive license meets another restrictive license**

  The second program with the restrictive license proves superior. The first program cannot improve, since the source code for the superior program is not available for examination; it cannot be used use for improvement.

  (That is to say, the developers of the worse program cannot learn from the source code of the better program. They cannot apply the lessons

that they learn from such an examination. Although the developers of the worse program may learn how to improve their package in other ways, they cannot learn by studying source code for the better program. Over time, the programs' users will tend to choose the better program because it better satisfies them.)

- **One between-style license meets another**

  The second program with the between-style license proves superior. In this case, the programmers for the first program can examine the code from the second and incorporate it. On the next encounter (remember that an evolutionarily stable strategy includes iteration), the two programs will enjoy more nearly equal performance.

  (That is to say, the users of the two programs will, over time, have less reason to select one over the other.)

- **GPL meets a between-style license**

  The program under the GPL proves superior. The program with the between-style can examine the code from the program under the GPL but cannot incorporate it without changing strategies.

I could continue. But rather than do that, let me summarize the outcome in a matrix.

```
First meets Second.
Benefit to the first,
depending on how both act.
```

[PayoEE to First]	Second Restrictive	Second Between	2nd GPL
First Restrictive	0	5	0
First Between	0	5	0
First GPL	0	5	5

If two programs with restrictive licenses meet, they gain nothing. If a program with a restrictive license meets a program with a between-style licence, the program with a restrictive license gains.

However, if a program with a restrictive license meets a program under the GPL, the program with the restrictive license gains nothing.

If two programs with between-style licenses meet, they each can gain from the other.

If a program under the GPL meets a program with a between-style license, the program under the GPL gains.

Finally, if two programs under the GPL meet, they each can gain from the other.

The most obvious point is that programs with a between-style strategy contribute both to programs with restrictive licenses and to programs under the GPL, as well as to each other.

As Rysdam says,

> *. . . in the terminology of game theory they are "suckers". The reason for this lies in the definition of [the between-style] strategy. All players can examine and use code from [between-style] strategy programs but [between-style] strategy programs can only use code from other [between-style] strategy programs. That is, [between-style] strategy programs cooperate well with each other, but do not keep "predators" from (ab)using them.*

However, no one can share code from programs with restrictive licenses, so they do not lose anything to competitors. At the same time, such programs cannot and do not help other.

Then, as Rysdam points out, programs under the GPL cooperate well because their sources are available to each other but they cannot be preyed on by programs under other licenses. In other words, programs under the GPL only cooperate with programs that are sure to cooperate back. They combine the best of all worlds.

This means that programs with a between-style strategy will tend to lose, leaving programs with either a restrictive license or a GNU General Public License.

After programs with a between-style strategy have lost, the matrix comes to look like this:

```
First meets Second.
Benefit to the first,
depending on how both act.
```

[Payoff to First]	Second Restrictive	2nd GPL
First Restrictive	0	0
First GPL	0	5

In this, the payoff is clear. Programmers who work on programs under a restrictive license cannot help other programmers who also work on programs under a restrictive license. Nor can they help programmers who work on programs under the GPL.

But a programmer who works on a program under the GPL can help other programmers who also work on programs under the GPL.

As David Rysdam wrote:

> *The GPL (and equivalent) licenses, when considered as strategies, are simply better adapted to the free market.*

12.3 Objections to the Theory

The analysis is straightforward. Nonetheless, people have raised objections to it. However, on further investigation, the objections fade.

Real licenses are not so simple

The first objection is that the licenses as used in the analysis are more simple than real licenses. In particular, the licenses in this analysis vary only by whether and under what conditions other developers may study and adapt programs' source code. Does not this simplification invalidate the result?

The response is simple: the double criterion is key. The history of the software industry tells us the criteria for success: whether programmers may study and adapt others' source code, and if so, under what conditions they may do so.

Obviously, a program with a restrictive license will benefit from programs that have a between-style license.

In one sense, this is no different from saying that a program with a restrictive license will benefit from more marketing on its behalf, or from customer 'lock-in'. People can be encouraged to stick to or adopt programs for a variety of reasons. However, using code 'invented elsewhere' may well be less expensive than locking in customers or paying for marketing.

In any event, a society does not advance the progress of software itself when it invests more in marketing a program, or in hindering consumer choice. But a society does advance the progress of software when programmers study, adapt, and use others' source code.

Clearly, the model is a simplification, but it is a telling one.

Programs differ

The second objection is that the analysis assumes that all programs are the same. Does the analysis fail to handle complexity? In reality, there are many different kinds of program, ranging from kernels to word processors to card games. This objection suggests, correctly, that the source code for a card game may not be all that much help to a kernel developer. It may be some help, but not as much help as the source for a different, but not too dissimilar kernel.

The solution is to apply the analysis a second and third time, once for programs that are kernels, a second time for programs that are word processors, and a third time for programs that are card games. In each case, the results are the same. Indeed, this games theory model is written so that it

works with any group of similar programs. It is not specific to one type of program.

The analysis is general.

People cheat

A more practical, and cynical, objection is that for the evolutionarily stable strategy to succeed, the programs under each of the different types of license must be counted on not to cheat, or, at least, not to cheat very much.

This is a valid objection. If programmers that develop code under a restrictive license cheat, they and their sponsors gain. This is also the case with crooks who rob banks and get away.

For this evolutionarily stable strategy to succeed, copyright holders and courts must enforce the law and deter crooks. Never coddle crooks. If crooks are not deterred, they must be caught and cured of their criminality.

Fortunately, over time, it is likely that crime will be discovered. As programs under the GPL become more successful and more widespread, more people will ask how programs under a restrictive license can continue to survive. The practices of their developers and sponsors will be subject to more and more scrutiny. And if they are found crooked, they will be caught. Crime will not pay.

History tells us . . .

Another question comes from a misunderstanding of history. The analysis tells us that GPL'd programs can survive against programs with restrictive licenses so long as governments support freedom. How then do we explain the growth of programs under restrictive licenses in the 1980's?

The answer is both simple and sad: in the 1980s, few programs were under the GPL or similar licenses. The programs under a non-restrictive license were mostly under a 'between-style' license. As expected, these programs were preyed upon; they provided help to programs under restrictive licenses, which grew in number and size.

Sources of income

The analysis provides an interesting insight into the economics implied by different style licenses:

Companies can employ police and courts to enable themselves to charge high fees for programs under restrictive licenses. You then can ask, will not these programs bring in more money to copyright holders than programs under a free license? The answer is long term, since this sort of change takes time, and is 'no'. The reason is that over the long term, developers who work on programs with restrictive licenses cannot help each other as much as developers who work on programs under the GPL. So the latter's programs will eventually become better.

Programs with restrictive licenses will eventually vanish. ('Eventually' may, of course, be a long time.) Companies cannot charge high fees for programs that no one wants. Instead, it is in companies' self-interest to change their strategies, both for licensing, and for making money. Companies are better off shifting to the GPL, and to making money by selling services or selling hardware. In the long run, companies are worse off when they depend on selling software at prices maintained by a government.

Cooperation

Yet another objection has to do with cooperation among software companies that restrict who may use their programs. Such companies often work together. This way, they gain the benefits of cooperation. The analysis dismisses this kind of cooperation.

Indeed, some companies cooperate with others, so their developers gain access to others' restricted code. However, such cooperation requires conscious action on the part of the managements of both companies. On the other hand, the GNU GPL automatically permits people to cooperate. Sharing is intrinsic.

The GPL thus lowers the cost of cooperation. As a consequence, there will be more of it.

Code Reuse

Another object has to do with code reuse. Everyone speaks of the value of reusing other's code. It is a clich\'e: 'don't reinvent the wheel'. But in practice, how much code is reused? Are not the ideas behind a program more important than the code itself?

It is true that programmers often write software a second time. But that also means that they and others must test the software a second time, and must implement fixes and advances a second time. It is more efficient to write and test once, to reuse code.

However, reuse works best with code that is modular and which follows standards that all may adopt. Free software programs tend to meet these criteria. For a restricted, proprietary program, there often is little reason in the short term for its programmers to make it modular; and a successful marketing monopoly provides a motivation against following free and international standards.

The lack of code reuse is a penalty paid more by those who work with restricted and proprietary programs than by those using free software.

13 Limits to Learning

Restrictions are not always immediately evident. If you are an end-user, you may not be aware of what you cannot get; it may not even occur to you that you have missed out on a development that did not take place.

If you are a programmer, you may find yourself limited in what you may study. Moreover, even if you have studied, you may be limited in what you may do with your learning. Either way, you are forced to act stupidly even though you can be smart.

In this chapter, I will primarily address programmers, since programmers do the work from which the rest of us benefit.

13.1 Trade Secrecy

Trade secrecy is a technique that is being employed in the U. S. and Europe to limit learning. In the U. S. and Europe, trade secrecy laws exist to protect companies from the inadvertent loss of a secret. The most famous example is the secret of the formula for Coca Cola. The Coca Cola company will not tell anyone how they make Coca Cola taste the way it does. The formula is a trade secret. If by some accident, the secret comes out, the courts will prevent anyone outside of the Coca Cola company from using the formula or telling people what it is.

In the past, a company was required to take due care and precaution to prevent the loss of a secret. If the secret was not well guarded, the courts would not protect it. The current legal campaign is to cause courts and police to guard secrets that are not protected.

This means, for example, that you might be forbidden to build and sell a mobile telephone that uses a communications protocol that is well known to you, but considered a 'trade secret'. Or, you might be forbidden to help a friend, or a customer, view a legally purchased movie on a legally purchased machine.

13.2 Ban Reverse Engineering

The second technique is to forbid you to study what others have done. If you cannot learn how to do something, you cannot do it.

Reverse engineering is the process by which you take apart a competitor's product and figure out what they did. You learn from them.

Very often, the original makers do not want others to study their work. But it has long been recognized that such study is good for an economy as a whole. Because of competitors, individual companies may not gain as much monopoly profit as they would like, but the competing businesses and the consuming public all benefit.

13.3 Patent Restrictions

Patent restrictions are a third way to limit you. A company will gain a patent on a mathematical area or business practice and then a government will provide enforcement. Unless you can invent around it, which is difficult if the patent covers a wide area mathematically, the patent will prevent you from working with free software.

For example, you may be interested in electronic business to business sales. B2B operations are efficient and potentially huge. They are a successful and useful way to use the Internet. In a free society, the best operations would succeed. But you may be hindered in trying to be best.

13.4 Trade-off between citizens' interests

In the United States of America, the purpose of copyright and patents is

> *To promote the progress of science and useful arts, . . .*
>
> `(U.S. Constitution, Article I, Section 8)`

That is to say, copyright and patents are a way a government influences an economy through bounty giving. Unlike direct government subsidies, however, a copyright or patent does not directly take from government revenues, except through the costs of policing and courts. Such costs are usually not attributed to specific patents or copyrights. Moreover, patent or copyright holders gain income only if others are considerably interested in the actions or information covered by the patent or copyright. Consequently, copyright and patents are, or were, thought to be among the better forms of government subsidy.

The idea behind both patents and copyrights was that new inventions, or new writings, are themselves common and inexpensive, but that the development and marketing of them is expensive. While individuals may come up with new inventions and new writings, only business organizations with considerable money could invest in the necessary development and marketing that makes inventions and writings worthwhile.

Hence, the U. S. Constitution would permit investors to establish monopolies on inventions or writings "for limited Times", and during that time to charge more for their manufacture than these products would in a competitive free market. The extra income to the investors would motivate them to develop and market the products. Moreover, the patent and copyright income would go only to those who sold products that considerably interested others; the products would be purchased only by people who would be willing to transfer more of their resources to an investor than would be considered 'fair' in a competitive, free market.

This was the trade-off: U. S. citizens would give up some of their rights to fair prices; in return, the government enforced restrictions would, it was thought, lead to more products becoming available than would have occurred otherwise.

The problem now is that patents and copyrights have become ever more widespread and their terms longer; and the nature of technology has changed. It is now far easier and cheaper to manufacture some kinds of product than ever before. So a trade-off that may well have made sense in the past loses its justification, which is to *. . . promote the progress of science and useful arts, . . .* (and not, as some have pretended, to provide a 'natural right' to copyright or patent holders).

In the United States, in so far as patents and copyrights fail their Constitutional purpose, they fail to have any justification in law. In every country, in so far as patents and copyrights fail to promote safety, quality, or opportunity, they fail to support civilization.

13.5 Different Attacks in Summary

These attacks on your ability to do business are based on proposed laws.

Each of the different limits to learning that I have mentioned:

- trade secrecy
- ban reverse engineering
- patent restrictions

either raises your costs but not the costs of your competitors or restricts what you can do.

Each of these is a different way for others to gain power over you.

14 Tiger teams and Poodle Teams

Now let us shift our attention to businesses that do not want to be robbed.

Let's look at the consequence of restrictions, a consequence that leads to more crime.

You have heard of "tiger teams". These are teams — often from military special forces — who attempt to break into secure sites, such as nuclear power plants, to test security.

You may not have heard of "poodle teams". Poodle teams are supposed to look like tiger teams, but are made up of poodles. When they think you might feed them, poodles will lick your hand.

Salesmen love poodle teams. They bark and growl when someone tries to take away their food. When disguised as tigers, they look ferocious.

Those who manage security at nuclear power plants, and other such places, hate tiger teams. Sometimes a tiger team succeeds in getting in. There is only one group that security managers hate more, real enemies.

It used to be that sites were vulnerable only to physical attack: but now many sites operate electronically and are on the Internet. These newfangled sites are like a building in the downtown of a great city, an electronic city that is larger than any physical city on the planet, a city composed of both St. Petersburg, Florida and St. Petersburg, Russia, of both Haarlem in the Netherlands, and Harlem in New York City.

These new sites are not only vulnerable to the old hazards of burglary, bribery, blackmail, bamboozlement, and misplaced belief, they are vulnerable to electronic attack.

Those who manage these sites need to test their defenses with tiger teams.

But sometimes they are offered poodle teams.

Poodle teams do not test security, but merely the appearance of it.

Crooks will find out the difference between appearance and reality; if the reality is weakness, crooks will rob.

14.1 Telling the difference

How does a security manager tell the difference between a tiger team and a poodle team? The answer is straightforward, simple, and politically sensitive. Ask that most cynical of questions, ask 'who benefits?'

Ask if the team might in any way benefit, directly or indirectly, by licking your hand, by telling you falsely that your security is good. If so, you are being offered a poodle team.

On the other hand, if the team — or its sponsors, or employers — benefit when they succeed in breaking your defenses, then you have a tiger team.

And how do you tell who benefits? For a poodle team, it is sufficient that a sponsor, or employer, must ask for permission *from someone other*

than you before attacking. If a team must ask someone else for permission to study source code, the team is full of poodles.

Now, of course, a salesman will tell you that getting such permission is a 'mere formality'. He will say that no one will ever have trouble getting permission to study sources. But that salesman will be lying to you. In fact, a vendor will provide such permission only to those who will not cause too much damage. (Being good salesmen, they will be quite happy showing off minor problems.) No vendor will voluntarily risk a great loss.

The members of a poodle team, however indirect the connection, will know who enables them to feed. You may think they are ferocious. Doubtless, they will growl convincingly, but their job is to lick your hand.

The tiger team is different. The members of a tiger team do not require permission from a vendor, or a vendor's agent, directly or indirectly. They have the freedom to study software for vulnerabilities. And they know they will never be punished, not three years from now, not fifteen years from now, for finding a vulnerability.

On the contrary, a tiger team is motivated to find problems. Their motivation is allied with yours, at least in the long run, however much their discoveries may pain you in the short run.

What is the necessary condition for a tiger team? Freedom: freedom to study source code. If you use software that is not free, you must worry lest your evaluation team is a poodle team, since their long-term motivation is for them to continue, for them to avoid their being hindered or banned. Their vulnerability takes away their freedom, and leaves you vulnerable.

You will hear wonderful stories by those who advocate poodle teams. They will tell you that you should purchase restricted and secret software.

But ask yourself, 'who, in the long run benefits and who loses, from the discovery of a vulnerability?' Unless you benefit, regardless of who else loses, you will be the loser.

As a practical matter of security, you need software freedom.

Otherwise, you will be licked by a poodle, and eaten by someone else's tiger.

15 The Manufacturing Delusion

Consider a business that sells software.

The idea that you should sell software **itself** is a business model that is a 'manufacturing delusion'. It is a decision to operate a business as if the software you distribute is similar to shoes or trucks. Earlier, I described the belief as a mistaken mental model. Nonetheless, given police support, as in the United States, companies can follow this business model. It is a mistake.

Software is not like a shoe or truck that is manufactured and then sold.

Firstly, perhaps 3/4 of the costs for a typical software package come *after* the software is first released. These are the costs of adapting existing software to new hardware, the costs of debugging it, and the costs of extending the software to handle new tasks.

A person who obtains a computer program does not want just the original, as with a pair of shoes or a truck. The user wants the debugged versions, the extended versions. When a company sells a truck for $25,000, it expects to spend several thousand dollars on warranty payments. But it does not expect to spend $75,000 on the truck, three times the money it received,

But this is what the 'Manufacturing Delusion' says to do with software: sell the package at a high initial price, and then provide the fixes and improvements at little or no additional cost. This leads to disaster.

For one, the owners of the software company see that fixes and improvements cost them money, rather than generate revenue. So they cut back on fixes and improvements. Instead, they encourage their staff to focus on initial sales to generate revenue. But existing customers then become upset and move to a competitor who offers a similar product that is better.

And since it is cheap to manufacture new copies of software, a competing company will reduce its prices to attract people to it.

Customers will only stick to one company if they feel they have no choice: they will stay only if they see that the cost of changing is higher than the cost of staying. This means that to become successful, a company that sells restricted software must become a monopolist. This is the nature of the situation.

That company must drive everyone else out of business, or at least, drive enough competitors out of business that the majority of its customers feel they have no practical choice of vendors. The successful company must make sure that no one else manufactures CDs with its software on it. So the successful monopolist must persuade its government to use its courts and police and foreign negotiators to prevent what it will call, dramatically, 'software piracy'.

The alternative is to work towards a different way of doing business, a different way of making a living — to work towards a competitive and free market.

15.1 Why Enter the Software Industry?

Because competition in a competitive market forces down the price of free software, no one should enter the software industry to sell software as such. Instead, a business should enter the industry to make money in other ways.

In a free software industry, companies and people do not sell software itself. For example, manufacturers sell CDs with software on it and the prices for these CDs are reasonable. Other companies sell services associated with software or they sell hardware or other solutions.

What services do I mean? Most directly, I mean help in using a computer, or, to take more specific examples, help in setting up a packet radio network, or help in creating and nurturing a warehouse data base.

Less directly, and increasingly, hardware companies that sell telephones or desalinization plants, add software to their products to make them more attractive to buyers.

And, of course, hardware companies sell hardware.

The profit-oriented case for free software revolves around economists' notion of a "complementary product".

As a general rule, and all else being equal, demand for a product increases when the price of its complements decrease. Thus, if the price of hotel rooms in Miami, FL, decreases, the number of airline passengers flying to Miami should increase.

Likewise, if the costs of software for a computer go down, the sales of services or hardware associated with that software should go up.

Put another way, profit-seeking companies try to commoditize their products' complements.

Companies enter the software industry in order to lower the costs of software, so that they can then sell more of software's complements.

16 Business models

There are different ways to sell software's complements. Each of these is a "business model", that is, the outline of a way to run a business.

First, consider *paid-for training*. It goes without saying that a government could pay for this kind of education, and some do. But I am thinking here of education that people pay for privately.

Famously, private educational and training services provide quick profits for those who enter the business early. (Eventually, the ease of entry means that more and more enter the industry and profits decline.)

A second model is summarized by the phrase *'Give Away the Razor, Sell Razor Blades'*. This describes the business model that the Gillette company adopted a century ago for its razors. It did not quite give away the holder for its razor blades, but it sold them at a loss; and it made money by selling razor blades. And it still does. I myself have paid the Gillette company far more for the razor blades I have bought from them than for their razors.

In this case, a company provides the software and sells support. Just as a razor blade holder is complementary to a razor, software is complementary to the support.

Some free software companies, like Red Hat, use software as a *'Market Positioner'*: the software brings people to them to purchase their other services. This is a third business model

Commercial free software companies may also *'sell a brand'*: that is to say, the companies provide a trusted product. This depends on having a known and good reputation.

Companies can do this in two ways: one, quite obviously, is to sell a software distribution. Customers know the company selected the software and did a good job, so the customer does not have to do the work.

A second, more subtle way to sell a brand is to sell certification: to guarantee a person's competence or a product's quality. Certification becomes more important to a society the more that people deal with strangers. Certification tells you whether a person or a product has the characteristics claimed.

In addition to selling services, or selling a brand, or selling the value inherent in a complete system, businesses can sell other kinds of products. We call this fourth business model *'Selling an adjunct'*.

One kind of product is that which goes with or explain a program. For example, O'Reilly sells computer books.

Similarly, a computer manufacturer can build new hardware, or recondition old hardware, and load it with inexpensive, customized, free software.

'Widget Frosting' is the name of a fifth business model that is similar to 'Selling an Adjunct', except that the product sold is more important than the software.

In English, a widget is an unspecified, manufactured object. Frosting is what you put on a cake, to make it more tasty. 'Widget Frosting' is the process of making a manufactured object more desirable to customers.

If you sell an Ethernet card or other small bit of hardware, you want your product to operate everywhere. Otherwise, you are making your market smaller for no good reason. One way to expand your market is to make the software for it free; this way others can adapt and use your hardware on their equipment, gaining sales for you.

More grandly, IBM, a large corporation, found that some of its customers refused to buy bigger and more expensive computers from IBM, even though they needed the larger capacity. The customers were afraid that their existing software would not run on the bigger machines. So IBM has adopted GNU/Linux to its whole range of hardware from its smallest laptop to its largest mainframe.

As a result, an IBM salesman can say 'look, GNU/Linux runs on the machine you are using now; and it runs on this bigger machine. Your software will run, too. So you can buy the bigger machine safely.' IBM uses the software to sell its hardware.

I should mention that most software is not written for sale, and never had been. Many people do not realize this.

Instead, most software is written for use in other products, like airplanes or ships, or in business or database systems. On its own, none of this software has what might be called a 'sale value'; it has only a 'use value'.

In the United States, less than 10% of all software is written to be sold.

However, the software that most people think about is sold under the 'manufacturing delusion'. It is visible. People who see a PC often think of the software on it. People who see a car or truck seldom think of the software in it.

I have been talking previously about the kind of software that people often think of selling, if they suffer from the 'manufacturing delusion'. What I want to turn to now is software that quite obviously has a 'use value' but whose 'sale value' is more dubious.

Companies that manufacture trucks or washing machines or electric generating plants often use the 'Widget Frosting' business model, at least in part.

They create software that runs inside their products and thereby make their products more attractive than they would be otherwise. These companies create embedded software.

There are two reasons such companies adopt free software. First, free software provides the company with an existing, complex system that works. The companies need to do less work of their own. It costs them less. Second, the free software leads to better products, so customers like them more. So the companies sell more.

Of course, other companies can use the same software: you need to give people a reason to buy from you. Here is where virtue becomes profitable. People will buy from you if your hardware is better, or your service is better, or if they like you for some other reason.

If you are not well known, people will be more likely to risk buying from you if they know they can hire someone else to work on your product. You reduce your customers' risk by providing them with free software.

It is a paradoxical rule: if it is easy for your customer to leave you, your customer is more likely to stay.

17 Concluding Remarks

In conclusion, your opportunities depend on your legal and practical freedom to:

- use,
- copy,
- redistribute,
- study, and,
- modify software.

In addition to freedom, software under the GNU General Public License imposes a duty:

- If you redistribute fixes and extensions to work others have done, then you must pass on to others the same rights you received.

Freedom is Key.

Freedom leads to:

- collaboration
- access
- empowerment
- lower prices
- choice
- reliability
- efficiency
- security
- fewer barriers to entry
- fewer barriers to use
- more opportunity

Appendix A GNU General Public License

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software — to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary.

To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice

that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:
   a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any

associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.
10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN

OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and a brief idea of what it does.
Copyright (C) 19yy  name of author

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) 19yy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.
```

The hypothetical commands '`show w`' and '`show c`' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than '`show w`' and '`show c`'; they could even be mouse-clicks or menu items — whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the program
'Gnomovision' (which makes passes at compilers) written by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may

consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Appendix B GNU Free Documentation License

Version 1.1, March 2000

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any

mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. In any section entitled "Acknowledgments" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgments and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these

sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties — for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgments", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this

License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

Concept Index

About the Author

Robert J. Chassell has worked with GNU Emacs since 1985. He writes and edits, teaches Emacs and Emacs Lisp, and speaks throughout the world on software freedom. Chassell was a founding Director and Treasurer of the Free Software Foundation, Inc. He is co-author of the *Texinfo* manual, and has edited more than a dozen other books. He graduated from Cambridge University, in England. He has an abiding interest in social and economic history and flies his own airplane.