Making a Living with Free Software *

— Software Freedom: Rights, Duty, Metaphor, and Making a Living

Robert J. Chassell
GNU Project, Free Software Foundation

bob@gnu.org

There is more to making a living with free software than running a business. You need an infrastructure of institutions to protect you and an understanding of the metaphors that shape people's thinking.

In this essay, I will first explain the nature of software freedom: your rights and your duty.

Then I will talk about the ways people who are not programmers come to understand software: the use of metaphor. In the United States, for example, you often hear the phrase *'The Information Highway'*. This is a metaphor.

Businessmen and government officials often base their judgements on metaphor.

Then I will describe the history of free software. The past tells us what to watch for in the future, and how to bring success out of failure.

I will then tell you what freedom brings:

```
* reliability,
* efficiency,
* security,
* choice of vendors,
* low barriers to entry,
* the legal and practical right to start and
  operate a business.
```

Finally, I want to focus on the business models of free software.

The successful business models recognize that revenue related to software comes from providing one or other kind of service.

*This speech was given by Robert Chassell when he visited China in the August of 2000.

1 What is free software?

1.1 What is software?

When I speak of software, I am speaking both about the programs that run the computer, that is to say, the operating system, and about applications, such as electronic mail and other communications, spreadsheets, electronic commerce, writing tools, sending and receiving FAXes, Web site creation, engineering, research, mathematical computations, modeling, image manipulation, and networking.

And I am talking about applications that are embedded in a machine, applications that control a fuel injector, or operate a telephone, or control a washing machine.

1.2 What is free software?

Free software is software that you may copy, study, modify, and redistribute. These are freedoms. They are not intrinsic to the technology: there exists software that you are forbidden to copy, forbidden to study, forbidden to modify, and forbidden to redistribute.

If you have software and you are forbidden to use it, and if that ban is enforced, the software does you no good.

It is fairly straightforward to copy and distribute software that is hard to study and modify. This is often done with Microsoft products, even though such actions are banned.

You gain little from software that is hard to study and hard modify since you cannot learn from it and cannot improve it. Indeed, such software makes you a dependent who cannot learn or advance. In the short run, you benefit because you can run a computer; that is why people use such software. But in the long run, you are hurt because you are held back.

For success, you need to study and modify software (or hire people who do this), as well as copy and redistribute it; the only way to ensure such success is a proper legal and institutional framework that protects your rights.

In addition, free software under the **GNU General Public License** (the GNU GPL) imposes on you an obligation to the community. This duty is enforceable by law as well as being an obligation that others expect you to follow. Your duty requires you to redistribute fixes and extensions that you make to work that others have done before you.

This is not a technical or business issue:

what makes software free rather than imprisoned is the legal and institutional framework in which people work.

There are two reasons to prefer free software over restricted-distribution software.

The first and most important reason is that a free society is better than the alternative. Your freedom to create and use software is a part of living in a better society. You have the legal right to choose a business, to choose a vendor, to choose software, to share with others, and to collaborate.

A second reason is that free software, over time, tends to become more reliable, efficient, and secure. These reasons motivate those who otherwise do not care whether you have any rights.

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 users want.

As a practical matter, the key to the good use of software is to ensure freedom. In software, this leads to reliability, efficiency, and security, to lower prices, to collaboration, and fewer barriers to entry and use.

2 Legal framework, ancient differences

The key, as I said, is freedom, the legal right to copy, study, modify, and redistribute software. Rights generate freedom.

It is important to ensure all 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 four rights. Without them, you lose the social and technical benefits.

For example, in the United States in the 1980s, I wanted to use a typesetting computer program call TEX. However, that program was in a legal state called the *'public domain'*.

People often think of the 'public domain' as best of all. In this case, it meant that anyone could use the software.

However, the legal status of 'public domain' also meant that companies could make additions to the software that they kept from the public.

It meant that I could not typeset and print a book that a friend had created on his computer, since the company that extended the original program to work on his computer did not cooperate with the company that extended the program to work on my computer. So even though the initial program was available to the public, it lived without proper protection; and was thereby made useless.

Fortunately, others have taken that original program, and added to it in well protected ways, and I can now use it again.

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.

Since you must deal with people who are strangers, people with whom you have no connections, you must, if necessary, be able to resort to law. Most people are honest and moral, but not everyone.

Without law, there are no practical sanctions. And 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.

I have a question about ancient history. 2,000 years ago in the Han Dynasty, there were few lawyers in China. It was not an honored and independent profession. But there were lawyers at that time in the Roman Empire. Businesses hired lawyers to work for them to settle disputes in courts provided by the government. And people in Europe began to think that that was the way disputes ought to be settled.

My understanding is that the Han Dynasty also had judges and courts, and these worked well for the maintenance of Han civilization. But there were no lawyers for businesses to hire; and many businesses were unregistered. So businesses did not make use of the government's courts. They settled disputes in other ways. And people in China began to think that that was the way things ought to be.

Of course, there have been many changes in the last 2,000 years. The Han Dynasty is long gone, as is the Roman Empire.

But does anyone in China perceive the use of law and of licences as a strange and perhaps foreign method of settling disputes?

I ask this because free software depends on the reuse of laws that were originally designed for other purposes. The GNU General Public License, for example, which I will talk about in more detail in a moment, is a copyright license. It depends on the law of copyright. And it depends on the proper use of honest courts and police.

If the ancient Roman Empire had had computers and copyright law, the GNU General Public License could have been used 2,000 years ago, to protect free software.

But I do not know how free software would have been protected during the Han Dynasty, especially if members of the government did not all favor freedom.

3 GPL, rights, duty, in detail

In the GNU Project, we created a legal tool to protect and preserve free software.

The legal tool is a specially drafted copyright license, the GNU General Public License.

In essence, this license forbids you to forbid. It also forbids others from preventing you from acting.

In addition, the GPL imposes on you an obligation. This duty is enforceable by law as well as being an obligation that others expect you to follow. Your duty requires you to redistribute source code that you create as an extension to work that others have done.

Before talking about your duties, let me first go through the list of rights that comes with free software: your rights to copy, study, modify, and redistribute the software.

The GNU General Public License gives you more rights than the usual copyright license. For this reason, the GPL is sometimes called a *'copyleft'*. This neologism depends on the multiple meanings of the word *'right'*.

Do the words 'right' and 'left' in Chinese, 'you' and 'zuo' in Pinyin, convey the same joke or pun as the words in English?

The four rights are to copy, study, modify, and redistribute.

First, the right to copy. Not many people own or manage a factory that would enable 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.

But everyone with access to a computer owns or manages a software factory, a device for manufacturing software, that is to say, for making new copies. Because copying software is so easy, we don't use the word 'manufacturing'; we usually do not even think of it as a kind of manufacturing, but it is.

The right to copy software is the right to use your property, your own means of production.

Second, the right to study. 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 lawyer to read legal text books. Unless you are in the profession, you probably wish to avoid such study.

However, this right to study has several implications, both for those who program and for everyone else.

The right to study means that people in places like Mexico,

or Germany, or China, 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 toy programs of school text books, no real programs.

Many years ago, a wise man said that the best way to see ahead and to advance is to sit on the shoulders of a giant.

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.

Software 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 a binary, but with great difficulty; it is seldom worth the effort. Source code is the best practical form for humans.)

The next right, the right to modify, is the right to fix a problem or enhance a program. 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.

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 China will use a program first written in Finland.

Finally, of these legal rights, comes the right to redistribute. This means that you, who own a computer, a software factory, have the right to make copies of a program and redistribute it. You can charge for these copies, or give them away. Others may do the same.

Remember, that redistributed code must include source code. Redistributed binary code lets you run a computer, but prevents you from doing anything else. It traps you in dependence.

When you create and distribute new code that fixes or extends older code, then you gain a duty, which is to distribute the sources for your new code, under the same license 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.

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

I mentioned earlier the typesetting computer program call TEX. This program was in the 'public domain', meaning it was not covered by a license. No license, no obligation: I described the problem this caused: I could not typeset a book.

However most programs do have licenses. But some licenses do not impose this obligation to redistribute fixes or extensions.

These other licenses, such as the famous BSD license, permit a person or company to take software that is itself free, and fix a bug or make an improvement, and then restrict who can 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 tax payer.

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, America On Line, 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 call it **'free software'**, many others refused to cooperate with Netscape. I myself think that this is one reason the new Netscape browser, the Mozilla project, is 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.

Now the new Netscape Web browser project, the Mozilla project, is finally coming to fruition, and the browser work well, but imagine if it had appeared a year or more ago!

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

And only through the law can we ensure that everyone acts upon their duty.

4 Competitive, free market collaboration

The right to redistribute, so long as it is defended and upheld, means that software is sold in a competitive, free market. This has several consequences. Low price is a consequence. This helps consumers.

But first and foremost, these legal and economic rights 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.

I will spend a few moments explaining this because it is important.

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, back in the United States, 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 have a semi-free market.

5 Lower, legal prices

Moreover, and this benefits people who are not programmers, if software is sold in a free market, competition among vendors will lead to a lower price. Put another way, the price of software is determined primarily by legal considerations: by the degree to which customers enjoy freedom.

If customers are forbidden to buy a product except at a high price, and that prohibition is successfully enforced, the product will be expensive. This is what occurs with much proprietary software today.

On the other hand, if software is sold in a free market, competition among vendors will lead to a lower price. This 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 supported 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. China is different. It was once considered a failure, but now various US companies are thinking it is a success. Hence, they are pressing the US government to persuade the Chinese government to adopt laws against illegal distribution. At some point, the Chinese government will have to enforce these laws to the satisfaction of companies like Microsoft, or else face trade sanctions.

On the other hand, free software is inexpensive and legal. It is more accessible. It is also customizable in ways that restricted software often is not. This is empowering.

As I said earlier, we shape the development of this technology, we create collaboration, through the use of a legal tool, a license, that gives you more rights than you would have otherwise, that forbids you to forbid, that in this case, gives you the right to copy, study, modify, and redistribute the software.

Because of the freedoms associated with it, this software is called *'free software.'*

While I am speaking of this phrase, let me clear up a verbal issue that sometimes confuses English speakers.

6 Meanings of word 'free'

The low price of free software leads some English speakers to think that the word 'free' in the phrase 'free software' means they can obtain it without cost. This is not the definition, which is about freedom, but it is an easy misunderstanding. After all, I have been talking of frugal use of resources, software that is inexpensive.

The English word 'free' has several meanings. As a Mexican friend of mine — and leader, by the way, of a major free software project — once said to me,

English is broken; it does not distinguish between 'free beer' and 'free speech'.

Spanish, on the other hand, distinguishes between *'gratis'* and *'libre'*. When you speak of 'free beer', you mean beer that is gratis; but when you speak of 'free speech' you mean freedom.

Free software is 'libre' software.

Incidentally, Eric Raymond and Bruce Perens invented the phrase 'open source' a few years ago as a synonym 'free software'. They wanted to work around the dislike many companies have of free markets. The phrase is popular; Eric and Bruce succeeded in their purpose.

However, I prefer the term 'free software' since it better conveys the goal of freedom; the proposition that every man and woman has the right to do first rate work, and must not be forbidden from doing so.

7 Metaphors explain the new in terms of the old

What is this technology to which people have rights, or should have rights? Let me explore this question in more detail by discussing how people think of one aspect of computers and software, which is the Internet.

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

In the United States, the most common metaphor for explaining the Internet is the phrase *'Information Highway'*. I don't know if this metaphor is so common in other parts of the world. If it is not common, it is still worth learning, because you will have to deal with Americans whose thoughts grow out of this metaphor, and are sometimes wrong.

The metaphor of the 'Information Highway' takes people's knowledge of highways and invites them to apply that knowledge to a new and for most people unknown artifact, the Internet.

In the Tang Dynasty, 1,500 years ago, I would have used the metaphor of the 'Information Canal' since the Grand Canal had just been extended, and many smaller canals built.

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 do need to gain access to the Internet, through a telephone, cable, 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 (or right next to a canal).

Moreover, the metaphor does not tell you that you can bring remote computers into your home or office. It did not warn me that that when I was 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.

Also, while the metaphor correctly tells you that Internet connections may be slow intrinsically, like a secondary road, or suffer traffic jams during rush hour, it misleadingly suggests that the system takes up a great deal of 'space' that could be used for other things, such as parks within a city.

It 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.

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 market 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, sweatheart deals, and cozy arrangements.

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 re-see the same information, just as many patrons can borrow the same book. This is important for those of you who concern yourself with budgets.

Moreover, the metaphor tells you to expect a vast range of queries; that 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.

Most importantly, aside from the pleasure a library gives, a great library enables people to learn from, and possibly avoid, the mistakes of others. Lessons learned: you do not have to repeat others failures; you can perhaps succeed!

Embedded software is software that goes into a washing machine or airplane or truck. It is not visible the way the Internet is visible. And while I am focusing mostly on the Internet, let's shift our attention for a moment.

People often use a metaphor to understand embedded software. Most often, the metaphor is of a little man, sitting in the washing machine or automobile engine, pulling levers.

The metaphor is valuable in that it tells you that the machine can respond to new conditions in ways it could not respond before. The metaphor is dangerous in that it can lead you to believe that the machine can do as much as a human, when it cannot.

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

To return again to the metaphor of the 'Information Highway': this metaphor 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.

As a practical matter, free software brings you each of these features: reliability, security, and efficiency.

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 must be good. It tells us about issues of privacy, and the opportunities for monopoly.

Freedom brings security, it brings trusted ways of dealing with one another, it brings the possibility of privacy, and it brings the makings 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.

Freedom in software, the right to copy, study, modify, and redistribute, brings you the results. They flow from technology, as shaped by the appropriate free license.

8 History

Next, I want to talk briefly about the history of free software. How did we get here?

This is not ancient history; it is recent. It is important, because it explains how we came to invent the GNU General Public License, how freedom meant success rather than failure, and how 'Linux' appeared.

Originally, all software was free. That is to say, programmers had the legal 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 was not onerous.

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.

In the early and mid 1980s, these hindrances inspired Richard Stallman and others (including me) to start GNU, a project to create an open source, freely redistributable operating system and associated applications.

We were able to do this because we had the legal right to invent a license, the GPL. Under the law, if you use our work, or if you fix or extend our work, you have to abide by the license conditions we set. The conditions we set are specified in the GNU General Public License, which gives you the rights to copy, study, modify, and redistribute the software, and the obligation to redistribute your changes under the same license (if you distribute the changes at all; if you keep your changes to yourself, you do not have to redistribute sources).

Of course, if you do not like the terms of our license, you do not have to use our software.

In the early 1990s, the main parts of the GNU Project were complete. We had written most of the necessary software.

However, work on a a key piece was delayed. FSF was developing a highly advanced operating system kernel. This is the software that schedules operations for the central process unit and does other important jobs.

Had this been a restricted-distribution project, the whole project would have failed, as so many have done, even though more than 7/8ths was completed, tested, and in use in other systems.

But this was a free software project, and Linus Torvalds, a young Finn, was able, legally and practically, to write his own, less advanced kernel. Linus called this kernel Linux, and adopted the GNU programs that were already written, the GNU environment. He also adopted the GNU General Public License, which made his contribution freely redistributable.

The combination of the GNU environment and the Linux kernel led to a usable operating system and set of applications called GNU/Linux, a name that is often shortened simply to Linux.

In the past couple of years, GNU/Linux has become widely known.

9 What freedom brings . . . in terms of software

Why is this technology successful? It is because of the benefits brought by freedom.

What does freedom bring?

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 entitities. 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 you can insert the wrong components, or leave them out. Such bugs cause havoc.

An advantage of free software is that lots of 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. He wants a good reputation.

In contrast, 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 the 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 what they are doing as less difficult than switching.

A notable feature of free software is that many applications run well on older, less capable machines. For example, a couple of months 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.

This frugality means that people can use older 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.

Moreover, free software brings with it frugal standards. You don't have to waste your correspondents' budgets by sending them overly bloated email.

A while back I received an email message that took up more than four and a half times the resources needed to convey the information.

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.

Your work should be secure. Your computer should avoid what you do not want.

Just recently, for example, 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.

You should have confidence in your privacy.

Of course, the free software producers don't always succeed, but on the whole, they have done well.

10 What freedom brings ... to customers and businesses

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 vendors also. Yes, it is easier for a customer to leave.

But this also means that customers are not frightened of working with a small business that they like, but figure may vanish in five or ten years; they can move without trouble. But I have also heard the opposite: the customer who decides to avoid a business because moving from it would be expensive, and the customer fears that the business will vanish in ten years.

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, too, since they too want to know they are living morally.

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.

Likewise, as a customer, you may use the code.

Freedom means that businesses are rewarded, with sales and profits, for satisfying customers legally, rather than rewarded by overcharging and hurting customers, which is illegal, at least in the US.

A quick digression here: restricted software often means 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 it.

Since free software is sold in a competitive market, its price is low. This means no one sells software as such. Instead, they sell services or they sell hardware as, for example, IBM does.

Success depends on satisfying your customers. This makes both your employees and your customers more happy.

The alternative is policing, which is to say, making sure that sotware is not used or copied illegally. Generally speaking, the word 'policing' is not used. Instead you hear of 'License Compliance' or other such phrase. A while back, the company that supplies me with electricity hired a 'License Compliance Manager' to make sure that engineers did not take their work home, since their work was associated with software that was not supposed to go out of the building. Policing is expensive and unpleasant.

First and formost, software freedom create a world in which software does what you want.

If you don't find an application that does what you want, you may write your own code, or hire someone to do so.

You have the legal right, and with the source code, the practical right, to adapt other code to what you want — this is often more efficient than writing from scratch.

Or, if you don't want to spend the money and resources, you can look around; often, you will find that someone else has faced nearly the same problem as you, and you can use that person's work.

11 What freedom brings . . . limitations

But freedom does not bring everthing on its own. Sometimes you cannot find a program that does what you want.

In particular, we need fully developed double entry booking software, for accounting.

When I first talked about accounting, free software for it did not exist. Now the software exists — I know of two packages — but they need more work.

Free software can be used any where in the world but we often do not see it used where it could be.

There are successes: inexpensive email in East Timor, a hospital using a free medical information management in Guatamala.

But often, you see people using tools that they are forbidden to study, learn from, modify, or customize. These packages, as I said earlier, solve one problem, but the user gains no other power from the software.

12 What freedom brings . . . ethical consequences

Free software permits legal sharing. This is an ethical issue. Do you want to encourage sharing? Should schools teach kids to be selfish, as required by the laws for restrict-

ed software?

As a practical matter, kids want to share. They want to help their friends. And as a practical and moral matter, everone wants others to be law abiding, even if they themselves are not.

So a government should arrange that being law abiding is best, for legal, moral, and practical reasons.

Let me return to freedom:

`Freedom brings the freedom to share. You have the legal right to help others. You have the legal right to collaborate. You can teach your children to share the software they have, legally.`

People who use binary-only software packages are forbidden to study them, learn from them, modify, or customize them. They gain no power from the software, except in so far as the package itself solves a problem.

Free software provides more than a solution; it provides the means for people to learn and become as good as or better than the programmers who wrote the software.

It empowers people who previously were kept out of the circle.

13 Schools

Before talking about business models, let me talk briefly about schools.

Students in school like to give copies of programs to their friends. Often, this giving is illegal. The programs' distribution is restricted and the school children are supposed to insist that their friends, or the school, purchase additional copies.

If you are a student or teacher or administrator in a school, you can spend a great deal of time trying to enforce the law.

Or you can teach your students to disobey the law. This common, but is a poor way to educate a society.

Even if people are not themselves law abiding, they always hope their neighbors will be law abiding and honest.

The solution is to adopt free software. Then you can encourage your students to give copies to each other: you can encourage them both to abide by the law and to share with others.

And of course, you can encourage the students to study the software that they have.

Students can learn to program, to maintain systems, and they can learn to learn, which is very important in a changing world.

14 Advantages to business

Now I would like to talk about the business of free software.

First, some background;

Free software imposes no legal barriers on the use of software. Free markets mean that software is inexpensive. Combined, these factors mean that free software reduces the 'barrier to entry' that often halts or prevents people from going into a new business.

In addition to reducing barriers to entry, free software reduces costs of operation. For example, as I said earlier, a good many of my correspondents pay telecommunications costs by the minute. They (or their supporting institutions) pay more for longer messages than for shorter ones. They prefer that I send email as plain text, which is not only an international standard, but conserves their resources.

As a managerial matter, a free software business is easier to run than a restricted distribution business.

Firstly, free software requires less policing than proprietary, restricted code since the product is sold in a competitive, free market where artificial retraints are not required. You do not have to lobby your government to police software pricing; the market does that for you.

You do, of course, have to persuade your government to enforce the free software license. Otherwise, people will take your work and give you nothing in return. They will fail their obligations to you.

Secondly, free software requires that your business focus on selling solutions to customers rather than on policing them.

This means that you can simplify your communications to your employees. You can tell them that their job is to create solutions to sell.

Otherwise, you have to tell them the rather more complex and mixed message that their jobs are both to create solutions to sell, and also, to enforce policing so that that some people who want to use your solutions are prevented from doing so.

15 The 'manufacturing delusion'

Now let me discuss a delusion — a delusion that I hope you avoid.

The idea that you should sell software itself is the 'manufacturing delusion'. This is a business model. It is a decision to operate a business as if the software you distribute is similar to shoes or trucks. 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.

As a practical matter, perhaps 3/4 of the costs for a typical software package come after the software is first released. These are costs of 'maintenance': 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.

The 'manufacturing delusion' says to sell software at a high initial price, as if it were a truck or shoe, 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 a successful company must become a monopolist, and 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 choice of vendors.

Also, of course, a company must make sure that no one else manufactures CDs with its software on it. So the successful monopolist will persuade its government to use its courts and police and foreign negotiators to prevent what it will call, dramatically, *'software piracy'*.

In the United States, as I mentioned earlier, the company that sells me electricity hired a 'software license compliance manager' to make sure that its engineers did not take work home with them. If the engineers took work home, they would take the software. And if they took the software, the company would be liable in court for breaking their software license. It is cheaper for the company to pay for its own policing than to end up in court.

I pay for this policing when I pay for electricity.

The 'manufacturing delusion' leads to catastrophe for all except the successful monopolist.

16 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 hardly sell software itself — manufacturers sell CDs with software

on it, but prices are reasonable. Instead, software companies and people sell services associated with software or hardware or other solutions.

What services do I mean? Most directly, 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.

The most common software business is that of support: to introduce people to computers, teach them how to use computers, fix problems as they arise, customize the software to local conditions, and so on.

Free software distributions include programs for secure communication via email, publishing, browsing, budgeting, and the like.

Training and support includes these as well as helping people to manipulate images, serve Web pages, or run an e-commerce site. The Debian GNU/Linux free software distribution has over 4,000 packages.

17 Business models

Now let me briefly discuss several business models.

First, 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.

This is one of the things done by our sponsor, Ron's DataCom. It sells the value inherent in providing a complete software system.

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

Free software companies sell a brand, like EasyLinux or Red Hat — that is to say, the companies get paid for providing 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 to others that some other product is good.

FSF-CHINA, for example is considering this: students must pass examinations to gain certificates of competence.

Besides 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[1].

Similarly, a computer manufacturer can put together hardware, or recondition old machinery, and load it with inexpensive, customized, free software.

(Again, early entrants can make large profits, before the industry matures.)

'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 ojbect. 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.

[1] Unfortunately, most books of O'Reilly are proprietary, not free — **FSM**

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 saleman 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 here 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 — embedded software — and thereby make their products better than they would be otherwise.

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.

And, of course, 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. Your reduce your customers risk by providing them with free software. It is the paradoxical rule: if it is easy for your customer to leave you, your customer is more likely to stay.

Now I want to turn to business models that do not directly generate revenue, but which reduce costs.

Cost-sharing cuts costs.

Suppose you are setting up a Web site. Perhaps you want to advertise your firm or inform people of what you do;, perhaps you want to sell things. To succeed, you need reliability, you need speed, and you need customizability.

How are you going to get these?

You have four choices:

You can buy a restricted-distribution Web server. In this case, you are betting that the seller's goals match your own, that the seller has the technical competence to do your job well, and that the seller will provide customization hooks that satisfy you.

A second choice is to write your Web server. By doing this, you get exactly what you want.

A third choice is to join the Apache group. This group consists of various firms that realized that it was smarter to join their efforts together.

A fourth choice is to adopt the Apache server, but not join

the Apache group.

The first choice is increasingly unpopular. Restricted-distribution companies have not provided the services or tools that people want.

The second choice is more popular than you might expect. It turns out that Web servers are reasonably easy to create. I have read of several different ones created for special purposes. However, not many write their own.

World wide, the two most popular choices are the third and the fourth.

If you adopt an Apache server, but do not join the group, you gain many of the advantages of others' work, but you lose the advantage of being in the middle of the decision making process: you are an observer of what others do. (This is what I do myself, since I run my Apache Web server on my home machine only for my own use — mostly for testing.)

If you wish to be central, you join the Apache group.

Yet another model is to reduce the chance of being hurt.

Cisco is the classic example of a company that decided to free some of its software so as to reduce the risk that it would be hurt sometime in the future.

Cisco manufactures and sells networking equipment. It also has a large local network with printers on it. (I think it uses more than 10,000 printers.)

The company wanted software to run its printers efficiently. So it hired two programmers to do the job. And they did.

However, the two programmers realized that they might in the future leave Cisco and that the company would have difficulty maintaining the software. So they suggested to Cisco that the software be made free.

This way, the software would be used outside the company as well as inside it; and people outside it would learn to fix and improve it.

Meanwhile, Cisco would have no sale value to lose, since they do not sell this product. So they would gain protection against a possible future loss.

18 Conclusion

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

```
copy,
study,
modify, and
redistribute

software under a free license.

Freedom is key.
```

Freedom leads to:

```
collaboration
lower prices
reliability
efficiency
security
fewer barriers to entry
fewer barriers to use
more opportunities in business.
```

About the Author Robert J. Chassell was a founding Director and Treasurer of the Free Software Foundation, Inc. The FSF was founded to support the GNU Project which restarted the movement towards free software and open sources. The GNU/Linux operating system and associated applications are the outcome of these efforts by the Foundation.

Chassell writes and edits. He is the author of *"An Introduction to Programming in Emacs Lisp"*, co-author of the *"Texinfo"* manual, and an editor of more than a dozen other books. He graduated from Cambridge University, in England. He flies his own airplane, and has an abiding interest in social and economic history. He could be reached by email: bob@gnu.org