The Danger of Software Patents

GNU philosophy

This is an unedited transcript of the talk presented by Richard Stallman on 8 October 2009 at Victoria University of Wellington, in Wellington, New Zealand.

This transcript was originally published on http://gnu.org, in 2009.

This document is part of GNU philosophy, the GNU Project's exhaustive collection of articles and essays about free software and related matters.

The Danger of Software Patents

I'm most known for starting the free software movement and leading development of the GNU operating system—although most of the people who use the system mistakenly believe it's Linux and think it was started by somebody else a decade later. But I'm not going to be speaking about any of that today. I'm here to talk about a legal danger to all software developers, distributors, and users: the danger of patents—on computational ideas, computational techniques, an idea for something you can do on a computer.

Now, to understand this issue, the first thing you need to realize is that patent law has nothing to do with copyright law—they're totally different. Whatever you learn about one of them, you can be sure it doesn't apply to the other.

So, for example, any time a person makes a statement about "intellectual property," that's spreading confusion, because it's lumping together not only these two laws but also at least a dozen others. They're all different, and the result is any statement which purports to be about "intellectual property" is pure confusion—either the person making the statement is confused, or the person is trying to confuse others. But either way, whether it's accidental or malicious, it's confusion.

Protect yourself from this confusion by rejecting any statement which makes use of that term. The only way to make thoughtful comments and think clear thoughts about any one of these laws is to distinguish it first from all the others, and talk or think about one particular law, so that we can understand what it actually does and then form conclusions about it. So I'll be talking about patent law, and what happens in those countries which have allowed patent law to restrict software.

So, what does a patent do? A patent is an explicit, government-issued monopoly on using a certain idea. In the patent there's a part called the claims, which describe exactly what you're not allowed to do (although they're written in a way you probably can't understand). It's a struggle to figure out what those prohibitions actually mean, and they may go on for many pages of fine print.

So the patent typically lasts for 20 years, which is a fairly long time in our field. Twenty years ago there was no World Wide Web—a tremendous amount of the use of computers goes on in an area which wasn't even possible to propose 20 years ago. So of course everything that people do on it is something that's new since 20 years ago—at least in some aspect it is new. So if patents had been applied for we'd be prohibited from doing all of it, and we may be prohibited from doing all of it in countries that have been foolish enough to have such a policy.

Most of the time, when people describe the function of the patent system, they have a vested interest in the system. They may be patent lawyers, or they may work in the Patent Office, or they may be in the patent office of a megacorporation, so they want you to like the system.

The *Economist* once referred to the patent system as "a time-consuming lottery." If you've ever seen publicity for a lottery, you understand how it works: they dwell on the very unlikely probability of winning, and they don't talk about the overwhelming likelihood of losing. In this way, they intentionally and systematically present a biased picture of what's likely to happen to you, without actually lying about any particular fact.

It's the same way for the publicity for the patent system: they talk about what it's like to walk down the street with a patent in your pocket—or first of all, what it's like to get a patent, then what it's like to have a patent in your pocket, and every so often you can pull it out and point it at somebody and say, "Give me your money."

To compensate for their bias, I'm going to describe it from the other side, the victim side—what it's like for people who want to develop or distribute or run software. You have to worry that any day someone might walk up to you and point a patent at you and say, "Give me your money."

If you want to develop software in a country that allows software patents, and you want to work with patent law, what will you have to do?

You could try to make a list of all the ideas that one might be able to find in the program that you're about to write, aside from the fact that you don't know that when you start writing the program. [But] even after you finish writing the program you wouldn't be able to make such a list.

The reason is. . . in the process you conceived of it in one particular way—you've got a mental structure to apply to your design. And because of that, it will block you from seeing other structures that somebody might use to understand the same program—because you're not coming to it fresh; you already designed it with one structure in mind. Someone else who sees it for the first time might see a different structure, which involves different ideas, and it would be hard for you to see what those other ideas are. But nonetheless they're implemented in your program, and those patents could prohibit your program, if those ideas are patented.

For instance, suppose there were graphical-idea patents and you wanted to draw a square. Well, you would realize that if there was a patent on a bottom edge, it would prohibit your square. You could put "bottom edge" on the list of all ideas implemented in your drawing. But you might not realize that somebody else with a patent on bottom corners could sue you easily also, because he could take your drawing and turn it by 45 degrees. And now your square is like this, and it has a bottom corner.

So you couldn't make a list of all the ideas which, if patented, could prohibit your program.

What you might try to do is find out all the ideas that are patented that might be in your program. Now you can't do that actually, because patent applications are kept secret for at least 18 months; and the result is the Patent Office could be considering now whether to issue a patent, and they won't tell you. And this is not just an academic, theoretical possibility.

For instance, in 1984 the Compress program was written, a program for compressing files using the data compression algorithm, and at that time there was no patent on that algorithm for compressing files. The author got the algorithm from an article in a journal. That was when we thought that the purpose of computer science journals was to publish algorithms so people could use them.

He wrote this program, he released it, and in 1985 a patent was issued on that algorithm. But the patent holder was cunning and didn't immediately go around telling people to stop using it. The patent holder figured, "Let's let everybody dig their grave deeper." A few years later they started threatening people; it became clear we couldn't use Compress, so I asked for people to suggest other algorithms we could use for compressing files.

And somebody wrote and said, "I developed another data compression algorithm that works better, I've written a program, I'd like to give it to you." So we got ready to release it, and a week before it was ready to be released, I read in the *New York Times* weekly patent column, which I rarely saw—it's a couple of times a year I might see it—but just by luck I saw that someone had gotten a patent for "inventing a new method of compressing data." And so I said we had better look at this, and sure enough it covered the program we were about to release. But it could have been worse: the patent could have been issued a year later, or two years later, or three years later, or five years later.

Anyway, someone else came up with another, even better compression algorithm, which was used in the program gzip, and just about everybody who wanted to compress files switched to gzip, so it sounds like a happy ending. But you'll hear more later. It's not entirely so happy.

So, you can't find out about the patents that are being considered even though they may prohibit your work once they come out, but you can find out about the already issued patents. They're all published by the Patent Office. The problem is you can't read them all, because there are too many of them.

In the US I believe there are hundreds of thousands of software patents; keeping track of them would be a tremendous job. So you're going to have to search for relevant patents. And you'll find a lot of relevant patents, but you won't necessarily find them all.

For instance, in the 80s and 90s, there was a patent on "natural order recalculation" in spreadsheets. Somebody once asked me for a copy of it, so I looked in our computer file which lists the patent numbers. And then I pulled out the drawer to get the paper copy of this patent and xeroxed it and sent it to him. And when he got it, he said, "I think you sent me the wrong patent. This is something about compilers." So I thought maybe our file has the wrong number in it. I looked in it again, and sure enough it said, "A method for compiling formulas into object code." So I started to read it to see if it was indeed the wrong patent. I read the claims, and sure enough it was the natural order recalculation patent, but it didn't use those terms. It didn't use the term "spreadsheet." In fact, what the patent prohibited was dozens of different ways of implementing topological sort—all the ways they could think of. But I don't think it used the term "topological sort."

So if you were writing a spreadsheet and you tried to find relevant patents by searching, you might have found a lot of patents. But you wouldn't have found this one until you told somebody, "Oh, I'm working on a spreadsheet," and he said, "Oh, did you know those other companies that are making spreadsheets are getting sued?" Then you would have found out.

Well, you can't find all the patents by searching, but you can find a lot of them. And then you've got to figure out what they mean, which is hard, because patents are written in tortuous legal language which is very hard to understand the real meaning of. So you're going to have to spend a lot of time talking with an expensive lawyer explaining what you want to do in order to find out from the lawyer whether you're allowed to do it.

Even the patent holders often can't recognize just what their patents mean. For instance, there's somebody named Paul Heckel who released a program for displaying a lot of data on a small screen, and based on a couple of the ideas in that program he got a couple of patents.

I once tried to find a simple way to describe what claim 1 of one of those patents covered. I found that I couldn't find any simpler way of saying it than what was in the patent itself; and that sentence, I couldn't manage to keep it all in my mind at once, no matter how hard I tried.

And Heckel couldn't follow it either, because when he saw HyperCard, all he noticed was it was nothing like his program. It didn't occur to him that the way his patent was written it might prohibit HyperCard; but his lawyer had that idea, so he threatened Apple. And then he threatened Apple's customers, and eventually Apple made a settlement with him which is secret, so we don't know who really won. And this is just an illustration of how hard it is for anybody to understand what a patent does or doesn't prohibit.

In fact, I once gave this speech and Heckel was in the audience. And at this point he jumped up and said, "That's not true, I just didn't know the scope of my protection." And I said, "Yeah, that's what I said," at which point he sat down and that was the end of my experience being heckled by Heckel. If I had said no, he probably would have found a way to argue with me.

Anyway, after a long, expensive conversation with a lawyer, the lawyer will give you an answer like this:

> If you do something in this area, you're almost certain to lose a lawsuit; if you do something in this area, there's a considerable chance of losing a lawsuit; and if you really want to be safe you've got to stay out of this area. But there's a sizeable element of chance in the outcome of any lawsuit.

So now that you have clear, predictable rules for doing business, what are you actually going to do? Well, there are three things that you could do to deal with the issue of any particular patent. One is to avoid it, another is to get a license for it, and the third is to invalidate it. So I'll talk about these one by one.

First, there's the possibility of avoiding the patent, which means, don't implement what it prohibits. Of course, if it's hard to tell what it prohibits, it might be hard to tell what would suffice to avoid it.

A couple of years ago Kodak sued Sun [for] using a patent for something having to do with object-oriented programming, and Sun didn't think it was infringing that patent. But the court decided it was; and when other people look at that patent they haven't the faintest idea whether that decision was right or not. No one can tell what that patent does or doesn't cover, but Sun had to pay hundreds of millions of dollars because of violating a completely incomprehensible law.

Sometimes you can tell what you need to avoid, and sometimes what you need to avoid is an algorithm.

For instance, I saw a patent for something like the fast Fourier transform, but it ran twice as fast. Well, if the ordinary FFT is fast enough for your application then that's an easy way to avoid this other one. And most of the time that would work. Once in a while you might be trying to do something where it runs doing FFT all the time, and it's just barely fast enough using the faster algorithm. And then you can't avoid it, although maybe you could wait a couple of years for a faster computer. But that's going to be rare. Most of the time that patent will to be easy to avoid.

On the other hand, a patent on an algorithm may be impossible to avoid. Consider the LZW data compression algorithm. Well, as I explained, we found a better data compression algorithm, and everybody who wanted to compress files switched to the program gzip which used the better algorithm. And the reason is, if you just want to compress the file and uncompress it later, you can tell people to use this program to uncompress it; then you can use any program with any algorithm, and you only care how well it works.

But LZW is used for other things, too; for instance the PostScript language specifies operators for LZW compression and LZW uncompression. It's no use having another, better algorithm because it makes a different format of data. They're not interoperable. If you compress it with the gzip algorithm, you won't be able to uncompress it using LZW. So no matter how good your other algorithm is, and no matter what it is, it just doesn't enable you to implement PostScript according to the specs.

But I noticed that users rarely ask their printers to compress things. Generally the only thing they want their printers to do is to uncompress; and I also noticed that both of the patents on the LZW algorithm were written in such a way that if your system can only uncompress, it's not forbidden. These patents were written so that they covered compression, and they had other claims covering both compression and uncompression; but there was no claim covering only uncompression. So I realized that if we implement only the uncompression for LZW, we would be safe. And although it would not satisfy the specification, it would please the users sufficiently; it would do what they actually needed. So that's how we barely squeaked by avoiding the two patents.

Now there is gif format, for images. That uses the LZW algorithm also. It didn't take long for people to define another image format, called png, which stands for "Png's Not Gif." I think it uses the gzip algorithm. And we started saying to people, "Don't use gif format, it's dangerous. Switch to png." And the users said, "Well, maybe some day, but the browsers don't implement it yet," and the browser developers said, "We may implement it someday, but there's not much demand from users."

Well, it's pretty obvious what's going on—gif was a de facto standard. In effect, asking people to switch to a different format, instead of their de facto standard, is like asking everyone in New Zealand to speak Hungarian. People will say, "Well, yeah, I'll learn to speak it after everyone else does." And so we never succeeded in asking people to stop using gif, even though one of those patent holders was going around to operators of web sites, threatening to sue them unless they could prove that all of the gifs on the site were made with authorized, licensed software.

So gif was a dangerous trap for a large part of our community. We thought we had an alternative to gif format, namely jpeg, but then somebody said, "I was just looking through my portfolio of patents"—I think it was somebody that just bought patents and used them to threaten people—and he said, "and I found that one of them covers jpeg format."

Well, jpeg was not a de facto standard, it's an official standard, issued by a standards committee; and the committee had a lawyer too. Their lawyer said he didn't think that this patent actually covered jpeg format.

So who's right? Well, this patent holder sued a bunch of companies, and if there was a decision, it would have said who was right. But I haven't heard about a decision; I'm not sure if there ever was one. I think they settled, and the settlement is almost certainly secret, which means that it didn't tell us anything about who's right.

These are fairly lightweight cases: one patent on jpeg, two patents on the LZW algorithm used in gif. Now you might wonder how come there are two patents on the same algorithm? It's not supposed to happen, but it did. And the reason is that the patent examiners can't possibly take the time to study every pair of things they might need to study and compare, because they're not allowed to take that much time. And because algorithms are just mathematics, there's no way you can narrow down which applications and patents you need to compare.

You see, in physical engineering fields, they can use the physical nature of what's going on to narrow things down. For instance, in chemical engineering, they can say, "What are the substances going in? What are the substances coming out?" If two different [patent] applications are different in that way, then they're not the same process so you don't need to worry. But the same math can be represented in ways that can look very different, and until you study them both together, you don't realize they're talking about the same thing. And, because of this, it's quite common to see the same thing get patented multiple times [in software].

Remember that program that was killed by a patent before we released it? Well, that algorithm got patented twice also. In one little field we've seen it happen in two cases that we ran into—the same algorithm being patented twice. Well, I think my explanation tells you why that happens.

But one or two patents is a lightweight case. What about mpeg2, the video format? I saw a list of over 70 patents covering that, and the negotiations to arrange a way for somebody to license all those patents took longer than developing the standard itself. The jpeg committee wanted to develop a follow-on standard, and they gave up. They said there were too many patents; there was no way to do it.

Sometimes it's a feature that's patented, and the only way to avoid that patent is not to implement that feature. For instance, the users of the word processor Xywrite once got a downgrade in the mail, which removed a feature. The feature was that you could define a list of abbreviations. For instance, if you define "exp" as an abbreviation for "experiment," then if you type "exp-space" or "exp-comma," the "exp" would change automatically to "experiment."

Then somebody who had a patent on this feature threatened them, and they concluded that the only thing they could do was to take the feature out. And so they sent all the users a downgrade.

But they also contacted me, because my Emacs editor had a feature like that starting from the late 70s. And it was described in the Emacs manual, so they thought I might be able to help them invalidate that patent. Well, I'm happy to know I've had at least one patentable idea in my life, but I'm unhappy that someone else patented it.

Fortunately, in fact, that patent was eventually invalidated, and partly on the strength of the fact that I had published using it earlier. But in the meantime they had had to remove this feature.

Now, to remove one or two features may not be a disaster. But when you have to remove 50 features, you could do it, but people are likely to say, "This program's no good; it's missing all the features I want." So it may not be a solution. And sometimes a patent is so broad that it wipes out an entire field, like the patent on public-key encryption, which in fact put public-key encryption basically off limits for about ten years.

So that's the option of avoiding the patent—often possible, but sometimes not, and there's a limit to how many patents you can avoid.

What about the next possibility, of getting a license for the patent?

Well, the patent holder may not offer you a license. It's entirely up to him. He could say, "I just want to shut you down." I once got a letter from somebody whose family business was making casino games, which were of course computerized, and he had been threatened by a patent holder who wanted to make his business shut down. He sent me the patent. Claim 1 was something like "a network with a multiplicity of computers, in which each computer supports a multiplicity of games, and allows a multiplicity of game sessions at the same time."

Now, I'm sure in the 1980s there was a university that set up a room with a network of workstations, and each workstation had some kind of windowing facility. All they had to do was to install multiple games and it would be possible to display multiple game sessions at once. This is so trivial and uninteresting that nobody would have bothered to publish an article about doing it. No one would have been interested in publishing an article about doing it, but it was worth patenting it. If it had occurred to you that you could get a monopoly on this trivial thing, then you could shut down your competitors with it.

But why does the Patent Office issue so many patents that seem absurd and trivial to us?

It's not because the patent examiners are stupid, it's because they're following a system, and the system has rules, and the rules lead to this result.

You see, if somebody has made a machine that does something once, and somebody else designs a machine that will do the same thing, but N times, for us that's a `for`-loop, but for the Patent Office that's an invention. If there are machines that can do A, and there are machines that can do B, and somebody designs a machine that can do A or B, for us that's an `if-then-else` statement, but for the Patent Office that's an invention. So they have very low standards, and they follow those standards; and the result is patents that look absurd and trivial to us. Whether they're legally valid I can't say. But every programmer who sees them laughs.

In any case, I was unable to suggest anything he could do to help himself, and he had to shut down his business. But most patent holders will offer you a license. It's likely to be rather expensive.

But there are some software developers that find it particularly easy to get licenses, most of the time. Those are the megacorporations. In any field the megacorporations generally own about half the patents, and they cross-license each other, and they can make anybody else cross-license if he's really producing anything. The result is that they end up painlessly with licenses for almost all the patents.

IBM wrote an article in its house magazine, *Think* magazine—I think it's issue 5, 1990—about the benefit IBM got from its almost 9,000 US patents at the time (now it's up to 45,000 or more). They said that one of the benefits was that they collected money, but the main benefit, which they said was perhaps an order of magnitude greater, was "getting access to the patents of others," namely cross-licensing.

What this means is since IBM, with so many patents, can make almost everybody give them a cross-license, IBM avoids almost all the grief that the patent system would have inflicted on anybody else. So that's why IBM wants software patents. That's why the megacorporations in general want software patents, because they know that by cross-licensing, they will have a sort of exclusive club on top of a mountain peak. And all the rest of us will be down here, and there's no way we can get up there. You know, if you're a genius, you might start up a small company and get some patents, but you'll never get into IBM's league, no matter what you do.

Now a lot of companies tell their employees, "Get us patents so we can defend ourselves" and they mean, "use them to try to get cross-licensing," but it just doesn't work well. It's not an effective strategy if you've got a small number of patents.

Suppose you've got three patents. One points there, one points there, and one points there, and somebody over there points a patent at you. Well, your three patents don't help you at all, because none of them points at him. On the other hand, sooner or later, somebody in the company is going to notice that this patent is actually pointing at some people, and [the company] could threaten them and squeeze money out of them—never mind that those people didn't attack this company.

So if your employer says to you, "We need some patents to defend ourselves, so help us get patents," I recommend this response:

> Boss, I trust you and I'm sure you would only use those patents to defend the company if it's attacked. But I don't know who's going to be the CEO of this company in five years. For all I know, it might get acquired by Microsoft. So I really can't trust the company's word to only use these patents for defense unless I get it in writing. Please put it in writing that any patents I provide for the company will only be used for self-defense and collective security, and not for repression, and then I'll be able to get patents for the company with a clean conscience.

It would be most interesting to raise this not just in private with your boss, but also on the company's discussion list.

The other thing that could happen is that the company could fail and its assets could be auctioned off, including the patents; and the patents will be bought by someone who means to use them to do something nasty.

This cross-licensing practice is very important to understand, because this is what punctures the argument of the software patent advocates who say that software patents are needed to protect the starving genius. They give you a scenario which is a series of unlikelihoods.

So let's look at it. According to this scenario, there's a brilliant designer of whatever, who's been working for years by himself in his attic coming up with a better way to do whatever it is. And now that it's ready, he wants to start a business and mass-produce this thing; and because his idea is so good his company will inevitably succeed— except for one thing: the big companies will compete with him and take all his market the away. And because of this, his business will almost certainly fail, and then he will starve.

Well, let's look at all the unlikely assumptions here.

First of all, that he comes up with this idea working by himself. That's not very likely. In a high-tech field, most progress is made by people working in a field, doing things and talking with people in the field. But I wouldn't say it's impossible, not that one thing by itself.

But anyway the next supposition is that he's going to start a business and that it's going to succeed. Well, just because he's a brilliant engineer doesn't mean that he's any good at running a business. Most new businesses fail; more than 95 percent of them, I think, fail within a few years. So that's probably what's going to happen to him, no matter what.

Ok, let's assume that in addition to being a brilliant engineer who came up with something great by himself, he's also talented at running businesses. If he has a knack for running businesses, then maybe his business won't fail. After all, not all new businesses fail, there are a certain few that succeed. Well, if he understands business, then instead of trying to go head to head with large companies, he might try to do things that small companies are better at and have a better chance of succeeding. He might succeed. But let's suppose it fails anyway. If he's so brilliant and has a knack for running businesses, I'm sure he won't starve, because somebody will want to give him a job.

So a series of unlikelihoods—it's not a very plausible scenario. But let's look at it anyway.

Because where they go from there is to say the patent system will "protect" our starving genius, because he can get a patent on this technique. And then when IBM wants to compete with him, he says, "IBM, you can't compete with me, because I've got this patent," and IBM says, "Oh, no, not again!"

Well, here's what really happens.

IBM says, "Oh, how nice, you have a patent. Well, we have this patent, and this patent, and this patent, and this patent, and this patent, all of which cover other ideas implemented in your product, and if you think you can fight us on all those, we'll pull out some more. So let's sign a cross-license agreement, and that way nobody will get hurt." Now since we've assumed that our genius understands business, he's going to realize that he has no choice. He's going to sign the cross-license agreement, as just about everybody does when IBM demands it. And then this means that IBM will get "access" to his patent, meaning IBM would be free to compete with him just as if there were no patents, which means that the supposed benefit that they claim he would get by having this patent is not real. He won't get this benefit.

The patent might "protect" him from competition from you or me, but not from IBM—not from the very megacorporations which the scenario says are the threat to him. You know in advance that there's got to be a flaw in this reasoning when people who are lobbyists for megacorporations recommend a policy supposedly because it's going to protect their small competitors from them. If it really were going to do that, they wouldn't be in favor of it. But this explains why [software patents] won't do it.

Even IBM can't always do this, because there are companies that we refer to as patent trolls or patent parasites, and their only business is using patents to squeeze money out of people who really make something.

Patent lawyers tell us that it's really wonderful to have patents in your field, but they don't have patents in their field. There are no patents on how to send or write a threatening letter, no patents on how to file a lawsuit, and no patents on how to persuade a judge or jury, so even IBM can't make the patent trolls cross-license. But IBM figures, "Our competition will have to pay them too; this is just part of the cost of doing business, and we can live with it." IBM and the other megacorporations figure that the general dominion over all activity that they get from their patents is good for them, and paying off the trolls they can live with. So that's why they want software patents.

There are also certain software developers who find it particularly difficult to get a patent license, and those are the developers of free software. The reason is that the usual patent license has conditions we can't possibly fulfill, because usual patent licenses demand a payment per copy. But when software gives users the freedom to distribute and make more copies, we have no way to count the copies that exist.

If someone offered me a patent license for a payment of one-millionth of a dollar per copy, the total amount of money I'd have to pay maybe is in my pocket now. Maybe it's $50, but I don't know if it's $50, or $49, or what, because there's no way I can count the copies that people have made.

A patent holder doesn't have to demand a payment per copy; a patent holder could offer you a license for a single lump sum, but those lump sums tend to be big, like US$100,000.

And the reason that we've been able to develop so much freedom-respecting software is [that] we can develop software without money, but we can't pay a lot of money without money. If we're forced to pay for the privilege of writing software for the public, we won't be able to do it very much.

That's the possibility of getting a license for the patent. The other possibility is to invalidate the patent. If the country considers software patents to be basically valid, and allowed, the only question is whether that particular patent meets the criteria. It's only useful to go to court if you've got an argument to make that might prevail.

What would that argument be? You have to find evidence that, years ago, before the patent was applied for, people knew about the same idea. And you'd have to find things today that demonstrate that they knew about it publicly at that time. So the dice were cast years ago, and if they came up favorably for you, and if you can prove that fact today, then you have an argument to use to try to invalidate the patent. And it might work.

It might cost you a lot of money to go through this case, and as a result, a probably invalid patent is a very frightening weapon to be threatened with if you don't have a lot of money. There are people who can't afford to defend their rights—lots of them. The ones who can afford it are the exception.

These are the three things that you might be able to do about each patent that prohibits something in your program. The thing is, whether each one is possible depends on different details of the circumstances, so some of the time, none of them is possible; and when that happens, your project is dead.

But lawyers in most countries tell us, "Don't try to find the patents in advance," and the reason is that the penalty for infringement is bigger if you knew about the patent. So what they tell you is "Keep your eyes shut. Don't try to find out about the patents, just go blindly taking your design decisions, and hope."

And of course, with each single design decision, you probably don't step on a patent. Probably nothing happens to you. But there are so many steps you have to take to get across the minefield, it's very unlikely you will get through safely. And of course, the patent holders don't all show up at the same time, so you don't know how many there are going to be.

The patent holder of the natural order recalculation patent was demanding 5 percent of the gross sales of every spreadsheet. You could imagine paying for a few such licenses, but what happens when patent holder number 20 comes along, and wants you to pay out the last remaining 5 percent? And then what happens when patent holder number 21 comes along?

People in business say that this scenario is amusing but absurd, because your business would fail long before you got there. They told me that two or three such licenses would make your business fail. So you'd never get to 20. They show up one by one, so you never know how many more there are going to be.

Software patents are a mess. They're a mess for software developers, but in addition they're a restriction on every computer user because software patents restrict what you can do on your computer.

This is very different from patents, for instance, on automobile engines. These only restrict companies that make cars; they don't restrict you and me. But software patents do restrict you and me, and everybody who uses computers. So we can't think of them in purely economic terms; we can't judge this issue purely in economic terms. There's something more important at stake.

But even in economic terms, the system is self-defeating, because its purpose is supposed to be to promote progress. Supposedly by creating this artificial incentive for people to publish ideas, it's going to help the field progress. But all it does is the exact opposite, because the big job in software is not coming up with ideas, it's implementing thousands of ideas together in one program. And software patents obstruct that, so they're economically self-defeating.

And there's even economic research showing that this is so—showing how in a field with a lot of incremental innovation, a patent system can actually reduce investment in R & D. And of course, it also obstructs development in other ways. So even if we ignore the injustice of software patents, even if we were to look at it in the narrow economic terms that are usually proposed, it's still harmful.

People sometimes respond by saying that "People in other fields have been living with patents for decades, and they've gotten used to it, so why should you be an exception?"

Now, that question has an absurd assumption. It's like saying, "Other people get cancer, why shouldn't you?" I think every time someone doesn't get cancer, that's good, regardless of what happened to the others. That question is absurd because of its presupposition that somehow we all have a duty to suffer the harm done by patents.

But there is a sensible question buried inside it, and that sensible question is "What differences are there between various fields that might affect what is good or bad patent policy in those fields?"

There is an important basic difference between fields in regard to how many patents are likely to prohibit or cover parts of any one product.

Now we have a naive idea in our minds which I'm trying to get rid of, because it's not true. And it's that on any one product there is one patent, and that patent covers the overall design of that product. So if you design a new product, it can't be patented already, and you will have an opportunity to get "the patent" on that product.

That's not how things work. In the 1800s, maybe they did, but not now. In fact, fields fall on a spectrum of how many patents [there are] per product. The beginning of the spectrum is one, but no field is like that today; fields are at various places on this spectrum.

The field that's closest to that is pharmaceuticals. A few decades ago, there really was one patent per pharmaceutical, at least at any time, because the patent covered the entire chemical formula of that one particular substance. Back then, if you developed a new drug, you could be sure it wasn't already patented by somebody else and you could get the one patent on that drug.

But that's not how it works now. Now there are broader patents, so now you could develop a new drug, and you're not allowed to make it because somebody has a broader patent which covers it already.

And there might even be a few such patents covering your new drug simultaneously, but there won't be hundreds. The reason is, our ability to do biochemical engineering is so limited that nobody knows how to combine so many ideas to make something that's useful in medicine. If you can combine a couple of them you're doing pretty well at our level of knowledge. But other fields involve combining more ideas to make one thing.

At the other end of the spectrum is software, where we can combine more ideas into one usable design than anybody else, because our field is basically easier than all other fields. I'm presuming that the intelligence of people in our field is the same as that of people in physical engineering. It's not that we're fundamentally better than they are; it's that our field is fundamentally easier, because we're working with mathematics.

A program is made out of mathematical components, which have a definition, whereas physical objects don't have a definition. The matter does what it does, so through the perversity of matter, your design may not work the way it "should" have worked. And that's just tough. You can't say that the matter has a bug in it, and the physical universe should get fixed. [Whereas] we [programmers] can make a castle that rests on a mathematically thin line, and it stays up because nothing weighs anything.

There're so many complications you have to cope with in physical engineering that we don't have to worry about.

For instance, when I put an `if`-statement inside of a `while`-loop,

- I don't have to worry that if this `while`-loop repeats at the wrong rate, the `if`-statement might start to vibrate and it might resonate and crack;
- I don't have to worry that if it resonates much faster—you know, millions of times per second—that it might generate radio frequency signals that might induce wrong values in other parts of the program;
- I don't have to worry that corrosive fluids from the environment might seep in between the `if`-statement and the `while`-statement and start eating away at them until the signals don't pass anymore;
- I don't have to worry about how the heat generated by my `if`-statement is going to get out through the `while`-statement so that it doesn't make the `if`-statement burn out; and
- I don't have to worry about how I would take out the broken `if`-statement if it does crack, burn, or corrode, and replace it with another `if`-statement to make the program run again.

For that matter, I don't have to worry about how I'm going to insert the `if`-statement inside the `while`-statement every time I produce a copy of the program. I don't have to design a factory to make copies of my program, because there are various general commands that will make copies of anything.

If I want to make copies on CD, I just have to write a master; and there's one program I can [use to] make a master out of anything, write any data I want. I can make a master CD and write it and send it off to a factory, and they'll duplicate whatever I send them. I don't have to design a different factory for each thing I want to duplicate.

Very often with physical engineering you have to do that; you have to design products for manufacturability. Designing the factory may even be a bigger job than designing the product, and then you may have to spend millions of dollars to build the factory. So with all of this trouble, you're not going to be able to put together so many different ideas in one product and have it work.

A physical design with a million nonrepeating different design elements is a gigantic project. A program with a million different design elements, that's nothing. It's a few hundred thousand lines of code, and a few people will write that in a few years, so it's not a big deal. So the result is that the patent system weighs proportionately heavier on us than it does on people in any other field who are being held back by the perversity of matter.

A lawyer did a study of one particular large program, namely the kernel Linux, which is used together with the GNU operating system that I launched. This was five years ago now; he found 283 different US patents, each of which appeared to prohibit some computation done somewhere in the code of Linux. At the time I saw an article saying that Linux was 0.25 percent of the whole system. So by multiplying 300 by 400 we can estimate the number of patents that would prohibit something in the whole system as being around 100,000. This is a very rough estimate only, and no more accurate information is available, since trying to figure it out would be a gigantic task.

Now this lawyer did not publish the list of patents, because that would have endangered the developers of Linux the kernel, putting them in a position where the penalties if they were sued would be greater. He didn't want to hurt them; he wanted to demonstrate how bad this problem is, of patent gridlock.

Programmers can understand this immediately, but politicians usually don't know much about programming; they usually imagine that patents are basically much like copyrights, only somehow stronger. They imagine that since software developers are not endangered by the copyrights on their work, that they won't be endangered by the patents on their work either. They imagine that, since when you write a program you have the copyright, [therefore likewise] if you write a program you have the patents also. This is false—so how do we give them a clue what patents would really do? What they really do in countries like the US?

I find it's useful to make an analogy between software and symphonies. Here's why it's a good analogy.

A program or symphony combines many ideas. A symphony combines many musical ideas. But you can't just pick a bunch of ideas and say "Here's my combination of ideas, do you like it?" Because in order to make them work you have to implement them all. You can’t just pick musical ideas and list them and say, “Hey, how do you like this combination?” You can’t hear that [list]. You have to write notes which implement all these ideas together.

The hard task, the thing most of us wouldn’t be any good at, is writing all these notes to make the whole thing sound good. Sure, lots of us could pick musical ideas out of a list, but we wouldn’t know how to write a good-sounding symphony to implement those ideas. Only some of us have that talent. That’s the thing that limits you. I could probably invent a few musical ideas, but I wouldn’t know how to use them to any effect.

So imagine that it’s the 1700s, and the governments of Europe decide that they want to promote the progress of symphonic music by establishing a system of musical idea patents, so that any musical idea described in words could be patented.

For instance, using a particular sequence of notes as a motif could be patented, or a chord progression could be patented, or a rhythmic pattern could be patented, or using certain instruments by themselves could be patented, or a format of repetitions in a movement could be patented. Any sort of musical idea that could be described in words would have been patentable.

Now imagine that it’s 1800 and you’re Beethoven, and you want to write a symphony. You’re going to find it’s much harder to write a symphony you don’t get sued for than to write one that sounds good, because you have to thread your way around all the patents that exist. If you complained about this, the patent holders would say, “Oh, Beethoven, you’re just jealous because we had these ideas first. Why don’t you go and think of some ideas of your own?”

Now Beethoven had ideas of his own. The reason he’s considered a great composer is because of all of the new ideas that he had, and he actually used. And he knew how to use them in such a way that they would work, which was to combine them with lots of well-known ideas. He could put a few new ideas into a composition together with a lot of old and uncontroversial ideas. And the result was a piece that was controversial, but not so much so that people couldn’t get used to it.

To us, Beethoven’s music doesn’t sound controversial; I’m told it was, when it was new. But because he combined his new ideas with a lot of known ideas, he was able to give people a chance to stretch a certain amount. And they could, which is why to us those ideas sound just fine. But nobody, not even a Beethoven, is such a genius that he could reinvent music from zero, not using any of the well-known ideas, and make something that people would want to listen to. And nobody is such a genius he could reinvent computing from zero, not using any of the well-known ideas, and make something that people want to use.

When the technological context changes so frequently, you end up with a situation where what was done 20 years ago is totally inadequate. Twenty years ago there was no World Wide Web. So, sure, people did a lot of things with computers back then, but what they want to do today are things that work with the World Wide Web. And you can’t do that using only the ideas that were known 20 years ago. And I presume that the technological context will continue to change, creating fresh opportunities for somebody to get patents that give the shaft to the whole field.

Big companies can even do this themselves. For instance, a few years ago Microsoft decided to make a phony open standard for documents and to get it approved as a standard by corrupting the International Standards Organization, which they did. But they designed it using something that Microsoft had patented. Microsoft is big enough that it can start with a patent, design a format or protocol to use that patented idea (whether it's helpful or not), in such a way that there's no way to be compatible unless you use that same idea too. And then Microsoft can make that a de facto standard with or without help from corrupted standards bodies. Just by its weight it can push people into using that format, and that basically means that they get a stranglehold over the whole world. So we need to show the politicians what's really going on here. We need to show them why this is bad.

Now I've heard it said that the reason New Zealand is considering software patents is that one large company wants to be given some monopolies. To restrict everyone in the country so that one company will make more money is the absolute opposite of statesmanship.