The Danger of Software Patents

Speech by Richard Stallman at Cambridge University,
25 March 2002

You might have been familiar with my work on free software. This speech is not about that. This speech is about a way of misusing laws to make software development a dangerous activity. This is about what happens when patent law gets applied to the field of software.

It is not about patenting software. That is a very bad way, a misleading way, to describe it, because it is not a matter of patenting individual programs. If it were, it would make no difference, it would be basically harmless. Instead, it is about patenting ideas. Every patent covers some idea. Software patents are patents which cover software ideas, ideas which you would use in developing software. That is what makes them a dangerous obstacle to all software development.

You may have heard people using a misleading term, "Intellectual Property". This term, as you can see, is biased. It makes an assumption that whatever it is you are talking about, the way to treat it is as a kind of property, which is one among many alternatives. This term "Intellectual Property" pre-judges the most basic question in whatever area you are dealing with. This is not conducive to clear and open minded thinking.

There is an additional problem [in that term] which has nothing to do with the promotion of any one opinion: It gets in the way of understanding even the facts. The term "intellectual property" is a catch-all; it lumps together completely disparate areas of law such as copyrights and patents, which are completely different. Every detail is different. It also lumps together trademarks which are even more different, and various other things more or less commonly encountered. None of them has anything in common with any of the others. Their origins historically are completely separate; the laws were designed independently; they covered different areas of life and activities; the public policy issues they raise are completely unrelated. So, if you try to think about them by lumping them together, you are guaranteed to come to foolish conclusions. There is literally no sensible intelligent opinion you can have about "Intellectual Property". [So] if you want to think clearly, don't lump them together. Think about copyrights, and then think about patents. Learn about copyright law, and separately learn about patent law.

To give you some of the biggest differences between copyrights and patents; Copyrights cover the details of expression of a work. Copyrights don't cover any ideas. Patents only cover ideas and the use of ideas. Copyrights happen automatically. Patents are issued by a patent office in response to an application.

Patents cost a lot of money. They cost even more paying the lawyers to write the application than they cost to actually apply. It takes typically some years for the application to get considered, even though patent offices do an extremely sloppy job [of] considering [them].

Copyrights last tremendously long. In some cases they can last as long as 150 years. Patents last 20 years, which is long enough that you can outlive them but still quite long by the timescale of a field such as software. Think back about 20 years ago when the PC was a new thing. Imagine being constrained to develop software using only the ideas which were known in 1982.

Copyrights cover copying. If you write a novel which turns out to be word-for-word the same as Gone With The Wind, and you can prove you never saw Gone With The Wind, that would be a defence to any accusation of copyright infringement.

A patent is an absolute monopoly on the use of an idea. Even if you could prove you had the idea on your own, that would be entirely irrelevant if the idea is patented by somebody else.

I hope you will forget about copyrights for the rest of this talk, because this talk is about patents, and you should never lump together copyrights and patents. It is about your understanding of these legal issues. It is like what would happen in your understanding of practical chemistry if you confused water and ethanol.

When you hear people describe the patent system, they usually describe it from the point of view of somebody who is hoping to get a patent--what it would be like for you to get a patent, what it would be like for you to be walking down the street with a patent in your pocket, so that every so often you can pull it out and point it at somebody and say "Give me your money!"

There is a reason for this bias, which is that most of the people who will tell you about the patent system have a stake in it, so they want you like it. There is another reason: the patent system is a lot like a lottery, because only a tiny fraction of patents actually bring any benefit to those who actually hold the patents. In fact, 'The Economist' once compared it to a "time-consuming lottery." If you have seen ads for lotteries, they always invite you to think about winning. They don't invite you to think about losing, even though losing is far more likely. It is the same for ads for the patent system. They always invite you to think about being the one who wins.

To balance this bias, I am going to describe the patent system from the point of view of its victims--that is, from the point of view of somebody who wants to develop software but is forced to contend with a system of software patents that might result in getting sued.

So, what is the first thing you are going to do after you have had an idea of what kind of program you are going to write?

The first thing you might want to try to do with the patent system is find out what patents may cover the program you want to write. This is impossible.

The reason is that some of the patent applications that are pending are secret. After a certain amount of time they may get published, like 18 months. But that is plenty of time for you to write a program, and even release it, not knowing that there is going to be a patent and you are going to get sued.

This is not just academic. In 1984, the compress program was written, a program for data compression. At the time, there was no patent on the LZW compression algorithm which it used. Then in 1985, the U.S. issued a patent on this algorithm and over the next few years, those who distributed the compress program started getting threats.

There was no way that the author of compress could have realised that he was likely to get sued. All he did was use an idea he found in a journal, just as programmers had always done. He hadn't realised that you could no longer safely use ideas that you found in a journal.

Let's forget about that problem. The issued patents are published by the patent office, so you can find the whole long list of them and see exactly what they say. You couldn't actually read the whole list as there are too many of them. In the U.S., there are hundreds of thousands of software patents. There is no way you can keep track of what they are all about. You would have to try to search for relevant ones.

Some people say that should be easy in these modern days of computers.

You could search for key words and so-on. That one works to a certain extent. You will find some patents in the area. You won't necessarily find them all however.

For instance, there was a software patent (which may have expired by now) on natural order recalculation in spreadsheets. This means basically that when you make certain cells depend on other cells, it always recalculates everything after things it depends on, so that after one calculation, everything is up to date. The first spreadsheets did their recalculation top-down, so if you made a cell depend on a cell lower down, and you had a few such steps, you had to recalculate several times to get the new values to propagate upwards. (You were supposed to have things depend on cells above them.)

Then someone realised, why don't I do the recalculation so that everything gets recalculated after the things it depends on? This algorithm is called topological sorting. The first reference I could find to it is in 1963. The patent covered several dozen different different ways you could implement topological sorting.

But you wouldn't have found this patent by searching for "spreadsheet." You couldn't have found it by searching for "natural order" or "topological sort." It didn't have any of those terms in it. In fact, it was described as a method of "compiling formulas into object code." When I first saw it, I thought it was the wrong patent.

Let's suppose that you got a list of patents, so you want to see what you are not allowed to do. When you try studying these patents, you will discover they are very hard to understand, as they are written in tortuous legal language whose meaning is very hard to understand. The things patent offices say often don't mean what they seem to mean.

There was an Australian government study of the patent system in the 1980's. It concluded that aside from international pressure, there was no reason to have a patent system--It did no good for the public--and recommended abolishing it if not for international pressure. One of the things they cited was that engineers don't try reading patents to learn anything, as it is too hard to understand them. They quoted one engineer saying "I can't recognise my own inventions in patentese".

This is not just theoretical. Around 1990, a programmer named Paul Heckel sued Apple claiming that Hypercard infringed a couple of his patents. When he first saw Hypercard, he didn't think it had anything to do with his patents, with his "Inventions". It didn't look similar. When his lawyer told him that you could read the patents as covering part of Hypercard, he decided to attack Apple. When I gave a speech about this at Stanford, he was in the audience. He said, "That's not true, I just didn't understand the extent of my protection!" I said "Yes, that's what I said."

So, in fact, you will have to spend a lot of time talking with lawyers to figure out what these patents prohibit you from doing. Ultimately they are going to say something like this: "If you do something in here, you are sure to lose; if you do something here [sweeps out a larger area], there is a substantial chance of losing, and if you really want to be safe, stay out of this area [an even larger area]. And, by the way, there is a substantial element of chance in the outcome of any law suit".

Now, that you have a predictable terrain for doing business(!) what are you going to do? Well, there are three approaches you might try. Any of which is applicable in some cases.

They are 1)Avoiding the patent 2)Licensing the patent 3)Overturning the patent in court.

Let me describe these three approaches and what makes them workable or unworkable.

1) Avoiding the patent. That means don't use the idea that the patent covers. This can be easy or hard, depending on what that idea is.

In some cases, a feature is patented. Then you avoid the patent by not implementing that feature. Then it just matters how important is that feature.

In some cases, you can live without it. A while ago, the users of the word processor XyWrite got a downgrade in the mail. The downgrade removed a feature which allowed you to pre-define abbreviations. That [is,] when you typed an abbreviation followed by a punctuation character, it would immediately replace itself with by some expansion, so that you could define the abbreviation for some long phrase, type the abbreviation then the phrase will be in your document. They wrote to me about this because they knew the Emacs editor has a similar feature. In fact, it had it since the 70's. This was interesting as it showed me that I had at least one patentable idea in my life. I knew it was patentable because somebody else patented it afterward!

Actually they tried these various approaches. First they tried negotiating with the patent holder, who turned out not to negotiate in good faith. Then they looked at whether they could have a chance at overturning the patent. What they decided to do was to take out the feature. You can live without this feature. If the word processor lacks only this feature, maybe people will still use it. But as various features start getting hit, eventually you end up with a program people think is not very good and they are likely to reject it.

That is a rather narrow patent on a very specific feature. What do you do with the British Telecom patent on traversing hyperlinks together with dial-up access? Traversing hyperlinks is absolutely essential to a major use of computers these days. Dial up access is also essential. How do you do without this feature, which, by the way, isn't even one feature, it is really a combination of two just arbitrarily juxtaposed? It is rather like having a patent on a sofa and television in the same room.

Sometimes the idea that's patented will be so broad and basic that it basically rules out an entire field. For instance, the idea of Public Key Encryption which was patented in the U.S. The patent expired in 1997. Until then, it largely blocked the use of Public Key Encryption in the U.S. A number of programs which people started to develop got crushed--they were never really available because the patent holders threatened them. Then, one program got away, the program PGP which initially was released as free software. Apparently, the patent holders, by the time they got around to attacking, realised they might get too much bad publicity. So they imposed restrictions, making it for non-commercial use only, which meant it couldn't catch on too much. They greatly limited the use of Public Key Encryption for a decade or more. There was no way around that patent. There was nothing else you could do like that.

Sometimes a specific algorithm gets patented. For instance, there is a patent on an optimised version of the Fast Fourier Transform. It runs about twice as fast. You can avoid that by using an ordinary FFT in your program. That part of the program will take twice as long. Maybe that doesn't matter, maybe that is a small part of the program's running time. Maybe if it is twice as slow, you won't really notice. Or maybe your program won't run at all as it will take twice real time to do it's job. The effects vary.

In some cases, you can find a better algorithm. This may or may not do you any good. Because we couldn't use compress, we started looking for an alternative compression algorithm. Somebody wrote to us saying he had one; he had written a program and decided to contribute it to us. We were going to release it. Just by chance, I happened to see a copy of the New York Times, it happened to have the weekly patent column in it. I didn't see a copy of the Times more than once every few months. So I looked at it and it said someone had got a patent for "Inventing a new method of compressing data". I figured I better take a look at this patent. I got a copy and it turned out to cover the program that we were just a week away from releasing. That program died before it was born.

Later on we did find another algorithm which was unpatented. That became the program gzip, which is now effectively the de-facto standard for data compression. As an algorithm to use in a program for data compression, it was fine. Anyone who wanted to do data compression could use gzip instead of compress.

The same LZW compression algorithm was also used in image formats such as the GIF format. But there, because the job people wanted to do was not to just compress data but to make an image that people could display with their software, it turned out extremely hard to switch over to a different algorithm. We have not been able to do it in 10 years! Yes, people used the gzip algorithm to define another image format, once people started to be threatened with law suits for using GIF files. When we started saying to people stop using GIF files, switch over to this, people said "We can't switch. The browsers don't support the new format yet". The browser developers said "We're not in a hurry about this. After all, nobody is using this new file format".

In effect, society had so much inertia in the use of the GIF format, we have not been able to get people to switch. Essentially, the community's use of the GIF format is still pushing sites into using GIF format with the result that they are vulnerable to these threats.

In fact, the situation is even more bizarre. There are in fact two patents covering the LZW compression algorithm. The patent office couldn't even tell they were issuing two patents on the same thing, they couldn't keep track. There is a reason for this: it takes a while of study of these two patents to see that they really cover the same thing.

If they were patents on some chemical process, it would be much easier. You could see what substances were being used, what the inputs were, what the outputs were, which physical actions are being taken. No matter how they are described, you'd see what they were and then you would see they are similar. If something is purely mathematical, there are many ways of describing it, which are a lot more different. They are not superficially similar. You have to really understand them to see they are really talking about the same thing. The patent office doesn't have time. The U.S patent office as of a few years ago was spending on average 17 hours per patent. This is not long enough to think carefully about them, so of course they make mistakes like that. In fact, I told you about the program that died before it was born. That algorithm also had two programs issued for it in the U.S.   Apparently, it is not that unusual.

Avoiding the patents may be easy, may be impossible. It may be easy but it makes your program useless. It varies depending on the situation.

Here is another point I should mention: Sometimes a company or consortium can make a format or protocol the de-facto standard. Then, if that format or protocol is patented, that is a real disaster for you. There are even official standards that are restricted by patents.

There was a big political uproar last September when the World Wide Web consortium was proposing to start adopting standards that were covered by patents. The community objected so they reversed themselves. They went back to insisting that any patents had to be freely implementable by anyone and that the standards had to be free for anyone to implement. That is an interesting victory. I think that was the first time any standards body has made that decision. It is normal for standards bodies to be willing to put something in a standard which is restricted by patents and people are not allowed to go ahead and implement it freely. We need to go to other standards bodies and call on them to change their rules.

2) Licensing the patent. The second possibility instead of avoiding the patent is to get a license for the patent. This is not necessarily an option. The patent holder does not have to offer you a license, it is not required. 10 years ago, the League for Programming Freedom got a letter asking for help from somebody whose family business was making gambling machinery for casinos, and they used computers back then. He received a threat from another company that said, "We have a patent. You are not allowed to make these things. Shut down!"

I looked at this patent. It covered having a number of computers on a network for playing games such that each computer supported more than one game and allowed you to play more than one game at a time.

You will find patent office really thinks there is something brilliant about doing more than one of anything. They don't realise that in computer science, that's the most obvious way to generalise anything. You did it once, so now you can do it any number of times, you can make a subroutine. They think that if you do anything more than once, that somehow means you are brilliant and that nobody can possibly argue with you and that you have the right to boss them around.

Anyway, he was not offered a licence. He had to shut down. He couldn't even afford really to go to court. I would say that particular patent was an obvious idea. It is possible that a judge might have agreed, but we will never know because he could not afford to go to court.

However, a lot of patent holders do offer licenses. They often charge a lot of money for that though. The company licensing the natural order recalculation patent was demanding 5% of the gross sales of every spreadsheet in the U.S. I am told that was the cheap pre-lawsuit price--if you actually made them sue you and they won, they'd demand more. You might be able to afford that 5% for licensing this one patent, but what if you need to license 20 different patents to make the program? Then all the money you take in goes on patents. What if you need to license 21 patents?

People in business told me that practically speaking, 2 or 3 of them would make any business unfeasible.

There is a situation where licensing patents is a very good solution. That is if you are a multinational mega-corporation. Because these companies own a lot of patents, and they cross-license with each other. That way, they escape most of the harm that the patent system does and they only get the good.

IBM published an article in Think magazine--I believe it was issue No. 5 of 1990--on IBM's patent portfolio, which said that IBM got two kinds of benefit from it's 9000 U.S. patents. (I believe the number is larger today.) These were, first, collecting royalties and second, getting "access to the patents of others." They said that the latter benefit is an order of magnitude greater. So the benefit that IBM got from being allowed to use the ideas that were patented by others was

10 times the direct benefit IBM could get from licensing patents.

What does this really mean? What is the benefit that IBM gets from this "access to the patents of others"? It is basically the benefit of being excused from the trouble that the patent system can cause you. The patent system is like a lottery. What happens with any given patent could be nothing, could be a windfall for some patent holder or a disaster for someone else. But IBM being so big [that], for them, it averages out. They get to measure the average harm and good of the patent system. For them, the trouble of the patent system would have been 10 times the good.

I say "would have been" because IBM through cross-licensing avoids experiencing that trouble. That trouble is only potential, it doesn't really happen to them. But when they measure the benefits of avoiding that trouble, they estimate it as 10 times the value of the money they collect from their patents.

This phenomenon of cross-licensing refutes a common myth, the myth of the "starving genius", the myth that patents "protect" the "small inventor". (Those terms are propoganda terms. You shouldn't use them.)

The scenario is like this: Suppose there is a "brilliant" designer of whatever. Suppose he has spent "years" starving in the attic designing a new wonderful kind of whatever, and now wants to manufacture it, and isn't it a shame the big companies are going to go into competition with him, take away all the business, and he'll "starve".

I have to point out that people in high tech fields are not generally working on their own and that ideas don't come in a vacuum, they are based on ideas of others; and these people have pretty good chances of getting a job if they need to these days. So this scenario, the idea that a brilliant idea came from this brilliant [person] working alone is unrealistic, and the idea that he is in danger of starving is unrealistic. But it is conceivable that somebody could have an idea and this idea along with 100 or 200 other ideas can be the basis of making some kind of product, and that big companies might want to compete with him.

So let's see what happens if he tries to use a patent to stop them. He says "Oh No, IBM. You cannot compete with me. I've got this patent." IBM says, "Let's see. Let's look at your product. Hmmm. I've got this patent and this one and this one and this one and this one and this one, which parts of your product infringe. If you think you can fight against all of them in court, I will just go back and find some more. So, why don't you cross license with me?" And then this brilliant small inventor says "Well, OK, I'll cross license". So he can go back and make these wonderful whatever it is, but so can IBM. IBM gets access to his patent and gets the right to compete with him, which means this patent didn't "protect" him at all. The patent system doesn't really do that.

The mega-corporations avoid the harm of the patent system; they see mainly the good side. That is why they want software patents: they are the ones who will benefit from it. But if you are a small inventor or work for a small company, the small company will not be able to do this. Small companies cannot get enough patents to do this.

Any given patent is pointing in a certain direction. So if a small company has patents pointing there, there and there and somebody over there points a patent at them and says give me your money, they are helpless. IBM can do it [retaliate] because with 9000 patents, they are pointing everywhere; no matter where you are, there is probably an IBM patent pointing at you. So IBM can almost always make you cross license. Small companies can only occasionally make someone cross-license. They will say they want patents for defensive purposes but they won't get enough to be able to defend themselves.

There are cases where even IBM cannot make someone cross-license. That is when there is a company whose sole business is taking a patent and squeezing money out of people. The company that had the natural order recalculation patent was exactly such a company. Their sole business was to threaten to sue people and collect money from people who were really developing something.

There are no patents on legal procedures. I guess the lawyers understand what a pain it would be to have to deal with the patent system themselves. The result is that there is no way to get a patent to make that company cross license with you. So they go around squeezing everyone. But I guess companies like IBM figure that is part of the price of doing business so they can live with it.

So that is the possibility of licensing a patent, which may or may not be possible, and you may or may not be able to afford it

The third possibility 3)Overturning a patent in court. Supposedly, in order to be patented, something has to be new, useful and unobvious. That is the language used in the U.S.; I think other countries have other language which is pretty much equivalent to it. Of course, when the patent office gets into the game, they start interpreting "new" and "unobvious". "New" turns out to mean "We don't have it in our files", and "unobvious" tends to mean "unobvious to someone with an I.Q of 50".

Somebody who studies most of the software patents issued in the U.S., or at least he used to, I don't know if he can still keep up with them, said 90% of them wouldn't have passed the "Crystal City test", which meant if the people in the patent office went outside to the news stand and got some computer magazines, they would see that these ideas are already known.

The patent office does things that are so obviously foolish, you wouldn't even have to know the state of the art to see they are foolish. This is not limited to software. I once saw the famous mouse patent which was obtained after Harvard genetically engineered a mouse with a cancer-causing gene. The cancer-causing gene was already known, and was inserted using known techniques into an already existing strain of mouse. The patent they got covered inserting any cancer-causing gene into any kind of mammal using any method whatsoever. You don't have to know anything about genetic engineering to realize that is ridiculous. I am told that this "overclaiming" is normal practice, and that the U.S. patent office sometimes invites patent applicants to make their claims broader--basically, make the claims broader until you think they are running into something else which that's unambiguous prior art. See how much land grab in mental space you can get away with.

When programmers look at a lot of software patents, they say "this is ridiculously obvious!" Patent bureaucrats have all sorts of excuses to justify ignoring what programmers think. They say "Oh! but you have to consider it in terms of the way things were 10 or 20 years ago." Then they discovered that if they talk something to death then you can eventually lose your bearings. Anything can look unobvious if you tear it apart enough, analyse it enough. You simply lose all standard of obviousness, or at least lose the ability to justify any standard of obvious or unobvious. Then, of course, they describe the patent holders as brilliant inventors, all of them, therefore we can't question their entitlement to power over what we do. If you go to court, the judges are likely to be a little more stringent about what is obvious or not. But the problem is that it costs millions of dollars to do that.

I heard of one patent case, the defendant I remember was Qualcomm, and I believe the ruling was ultimately 13 million dollars of which most went to pay the lawyers on both sides. There were a few million dollars left over for the plaintiff (because they [Qualcomm] lost).

To a large extent, the question of the validity of a patent will depend on historical accidents. Lots of historical accidents such as precisely what was published when, and which of those things somebody manages to find. Which of them didn't get lost, precise dates and so-on. Many historical accidents determine whether a patent is valid.

In fact, it is a weird thing that the British Telecom following hyperlinks together with telephone access patent was applied for in 1975. I think it was in 1974 that I developed the Info package for the first time. The Info package allows you to traverse hyperlinks, and people did use telephones to dial up and access the system. So in fact, I did produce a piece of prior art for this patent. This is the second piece of prior art I have produced in my life.

I didn't think this was interesting enough to publish it. After all, the idea of following hyperlinks I got from the demo of Englebart's editor. He is the one who had an idea which was interesting to publish. What I done I called "poor man's hypertext", as I had to implement it in the context of TECO. It was not as powerful as his hypertext, but it was at least useful for browsing documentation, which it all it was meant for, and as for there being dial-up access to the system, well, there was, but it didn't occur to me that the one had anything particular to do with the other. I wasn't going to publish a paper saying, "Oh! I implemented this poor man's hypertext, and guess what! There are dial-up lines on the computer too!" I suspect there is no way to tell precisely on what date I implemented this.

And was it published in any sense? Well, we invited guests to come in across the ARPAnet, and log in on our machine, so they could have browsed documentation using Info and seen the thing. If they had asked us, they would have found we had dial-up access. As you can see, historical accident determines whether you have prior art.

Now of course, there is a publication made by Englebart about hypertext, which they [the defendants] are going to show. I don't think it says anything about having dial-ups on the computer however, so whether it will suffice is not clear.

So, this is an option, the possibility of going to court to overturn the patent. Because of the expense, it is often out of the question even if you can find solid prior art which ought to be sufficient to overturn the patent. As a result, an invalid patent, a patent which nominally shouldn't have existed (but in fact lots and lots of them do) is a dangerous weapon. If someone attacks you with an invalid patent, that can really cause a lot of trouble for you. You might be able to bluff them away by showing them the prior art. It depends on whether they can get scared off that way. They might think, "Well, you are just bluffing, we figure you can't really go to court, you can't afford it so we'll sue you anyway".

All of these three possibilities are things that sometimes you can manage to use, but often you can't. So you have to face patent after patent after patent. Each time you may be able to find one of these three possibilities you can use, then there is another patent, then another and another. It gets like crossing a minefield. Each step you take, each design decision, probably won't step on a patent, so you can take a few steps and probably there won't be an explosion. But the chance you can get all the way through the minefield and develop the program you want to develop without ever stepping on a patent gets less and less as the program gets bigger.

Now, people used to say to me, "Well, there are patents in other fields, why should software be exempt?" Note the bizarre assumption in there, that somehow we are all supposed to suffer through the patent system. It is like saying "Some people get cancer. Why should you be exempt?" As I see it, each person who doesn't get cancer is good.

But there is, behind that, a less biased question, a good question, which is: Is software different from other fields? Should patent policy be different in different fields? If so, why?

Let me address that question: patents relate to different fields differently because in different fields patents relate to products differently.

On one extreme we have pharmecuticals where a given chemical formula would be patented, so that patent covers one and only one product. Some other product wouldn't be covered by the existing patent. If there is to be a patent for this new product, the patent holder would be whoever developed the new product.

This fits in with the naive idea of the patent system that we have, that if you are designing a new product, you are going to get "the patent". The idea that there is one patent per product and that it covers the idea of the product. In some fields it is closer to being true. In other fields it is further from being true.

[The software field is at that extreme.] This is because software packages are usually very big. They use many different ideas in a new combination. If the program is new and not just copied, then it is probably using a different combination of ideas--combined, of course, with newly written code, because you can't just magically say the names of these ideas and have them work. You have to implement them all. You have to implement them all in that combination.

The result is that even when you write a program, you are using lots of different ideas, any one of them might be patented by somebody. A pair of them may be patented as a combination by somebody. There might be several different ways of describing one idea which might be patented by various different people. So there are possibly thousands of things, thousands of points of vulnerability in your program, which might be patented by somebody else already.

This is why software patents tend to obstruct the progress of software--the work of software development. If it were one patent-one product, then these patents wouldn't obstruct the development of products because if you develop a new product, it wouldn't be patented by somebody else already. But when one product corresponds to many different ideas combined, it gets very likely your new product is going to be patented by somebody else already.

In fact, there is economic research now showing just how imposing a patent system on a field where there is incremental innovation can retard progress. You see, the advocates of software patents say "Well, yes, there may be problems, but more important than any problems, the patents must promote innovation, and that is so important it doesn't matter what problems you cause". Of course, they don't say that out loud because it is ridiculous, but implicitly they want you to believe that as long as it promotes progress, that outweighs any possible cost. But actually, there is no reason to believe it does promote progress. We now have a model showing precisely how patents can retard progress. The case where that model can fit describes the software field pretty well; incremental innovation.

Why is software on that extreme of the spectrum? The reason is that in software we are developing idealised mathematical objects. You can build a complicated castle and have it rest on a thin line and it will

stay up because it doesn't weigh anything. In other fields, people
have to cope with the perversity of matter--of physical
objects. Matter does what it is going to do. You can try to model it,
[but] if the actual behaviour doesn't fit the model then tough on you,
because the challenge is to make physical objects that really work.

If I wanted to put an 'If' statement in a 'While' statement, I don't
have to worry about whether the 'If' statement will oscillate at a
certain frequency and rub against the 'While' statement and eventually
they will fracture. I don't have to worry whether it will oscillate at
a certain higher frequency and induce a signal in the value of some
other variable. I don't have to worry about how much current that 'If'
statement will draw, and whether it can dissipate the heat there inside
that 'While' statement. Whether there will be a voltage drop across the
'While' statement that will make the 'If' statement not function. I
don't have to worry that if I run this program in a salt water
environment, the salt water may get in between the 'If' statement
and the 'While' statement and cause corrosion.

I don't have to worry when I refer to the value of a variable whether
I am exceeding the fan-out limit by referring to it 20 times. I don't
have to worry how much capacitance it has and whether there has been
sufficient time to charge up the value. I don't have to worry when I
write the program, about how I am going to physically assemble each
copy and whether I can manage to get access to put that 'If' statement
inside the 'While' statement. I don't have to worry about how I am
going to gain access in case that 'If' statement breaks, to remove it
and replace it with a new one. [There are] so many problems that we
don't have to worry about in software. That makes it fundamentally
easier. It is fundamentally easier to write a program than to design
a physical object that's going to work.

This may seem strange because you have probably heard people talking
about how hard software is to design and how this is a big problem and
how we are going to solve it. They are not really talking about the
same question as I am. I am comparing physical and software systems
of the same complexity, the same number of parts. I am saying the
software system is much easier to design than the physical system.
But the intelligence of people in these various fields is the same, so
what do we do when we are confronted with an easy field? We push it
further! We push our abilities to the limit. If systems of the same
size are easy, let's make systems which are ten times as big, then it
will be hard! That's what we do: we make software systems which are
far bigger in terms of number of parts than physical systems.

A physical system whose design has a million different pieces in it is
a mega project. A computer program whose design has a million pieces
in it is maybe 300,000 lines; a few people will write that in a couple
of years. That is not a particularly giant program. GNU Emacs now
has several million pieces in its design, I think. It has a million
lines of code. This is a project done with essentially no funding
whatsoever, mostly done by people in their spare time.

There is another big saving. If you have designed a physical product,
the next thing you have to do is design the factory to make it. To
build this factory may cost millions or tens of millions, whereas to
make copies of the program you just have to type 'copy'. The same
copy command will copy any program. You want copies on CD, then fine,
You burn a master CD and send it off to a CD plant. They will use the
same equipment which will copy any contents on a CD. You don't have
to build a factory to make this product.

There is tremendous simplification and tremendous reduction in costs
of designing things. The result is, say for an automobile company,
who will spend 50 million dollars to build a factory, to build a new
model of auto, they can hire some lawyers to cope with patent license

negotiations. They can even cope with a law suit if they wanted to. To design a program of the same complexity may cost 50 thousand or 100 thousand dollars. By comparison, the cost of dealing with the patent system is crushing. Or actually designing a program with the same complexity as the mechanical design of an auto is probably a month's work. How many parts does an auto have…that is, if it is an auto which doesn't have computers in it. That is not to say designing a good one is easy, but just that there are not that many different things in it.

The result is software really is different from other fields because we are working with mathematical stuff designing something is far, far easier and the result is that we regularly make systems which are much, much larger and do so with just a few people. The result is that instead of being close to one product-one patent, we are in a system where one product involves many, many ideas which could be patented already.

The best way to explain it by analogy is with symphonies. A symphony is also long and has many notes in it, and probably uses many musical ideas. Imagine if the governments of Europe in the 1700's had decided they wanted to promote the progress of symphonic music by establishing a European musical patent office that would give patents for any kind of musical ideas which you could state in words. Then imagine it is around 1800 and you are Beethoven and you want to write a symphony. You will find that getting your symphony so that it doesn't infringe any patents is going to be harder than writing a good symphony.

When you complain about this, the patent holders would say "Aw Beethoven, you are just bitching because you have no ideas of your own. All you want to do is rip off our inventions". Beethoven, as it happens, had a lot of new musical ideas--ut he had to use a lot of existing musical ideas in order to make recognisable music, in order to make music that listeners could possibly like, that they could recognise as music. Nobody is so brilliant that he can re-invent music [completely different] and make something that people would want to listen to. Pierre Boulez said he would try to do that, but who listens to Pierre Boulez?

Nobody is so brilliant he can re-invent all of computer science, completely new. If he did, he would make something that the users would find so strange that they wouldn't want to use it. If you look at a word processor today, you would find, I think, hundreds of different features. If you develop a nice new innovative word processor, that means there are some new ideas in it, but there must be hundreds of old ideas in it. If you are not allowed to use them, you cannot make an innovative word processor. Because the work of software development is so big, the result is that we don't need any artificial scheme to incentivise new ideas. You just have people writing software and they will have some new ideas. If you want to write a program and you want to make it good, some ideas will come to you and some you will see a way to use.

What used to happen--because I was in the software field before there were software patents--was most of the developers would publish any new ideas that they thought were noteworthy, that they thought that they might get any credit or respect for. The ideas that were too small or not impressive enough, they would not publish because that would be silly. Now the patent system is supposed to encourage disclosure of ideas. In fact, in the old days, nobody kept the ideas secret. They kept the code secret, it's true. The code, after all, represented the bulk of the work. They would keep the code secret and publish the ideas so that way the employees would get some credit and feel good. After software [patents], they still kept the code secret and they patented the ideas, so in fact, disclosure has not been encouraged in any meaningful sense. The same things are kept secret now as what were kept secret before, but the ideas which used to be published so that we could use them are now likely to be patented and off-limits for 20 years.

What can a country do to change this? How should we change the policy to solve this problem?

There are two places you can attack it. One is the place where patents are being issued, in the patent office. The other is where patents are being applied. That is a question of what does a patent cover.

One way is to keep a good criterion for issuing patents. This can work in a country which has not authorised software patents before, for instance, for the most part, in Europe. Simply to clearly re-enforce the European Patent Office's rules which say that software is not patentable. This is a good solution for Europe. Europe is now considering a directive on software patents. (The directive I suppose may be broader than that, but one of its important implications is for software patents.) Simply by modifying this to say software ideas cannot be patented will keep the problem out of Europe for the most part, except for some countries that may have admitted the problem on their own. Unfortunately one of them being the U.K. (Unfortunately for you.)

That approach won't work in the U.S. The reason is that the U.S already has large numbers of software patents, and any change in the criteria for issuing patents won't get rid of the existing ones. In fact, these patents are not officially labeled as software patents. I say software patents but what do I really mean? Patents which might potentially apply to software, patents which might potentially get you sued for writing software. The patent office doesn't divide patents into software patents and other patents. So, in fact, any patent might conceivably get you sued for writing software if it could apply to some software. So, in the U.S., the solution would have to be done through changing the applicability, the scope, of patents, saying that a pure software implementation running on general purpose computer hardware which does not in itself infringe the patent is not covered by any patent, and you cannot get sued for it. That is the other kind of solution.

The first kind of solution, the solution that operates on what types of patents can be valid, is a good solution for Europe to use.

When the U.S. started having software patents, there was no political debate. In fact, nobody noticed. The software field, for the most part, didn't even notice. There was a Supreme Court decision in 1981 which considered a patent on a process for curing rubber. The ruling was that the fact that the apparatus included a computer and a program as part of the process to cure the rubber didn't make it unpatentable. The next year, the appeals court which considers all patent cases reversed the quantifiers: they said the fact that there is a computer and a program in this makes it patentable. The fact that there is a computer and program in anything makes it patentable. This is why the U.S started having business procedure patents: because the business procedures were carried out on a computer and that made them patentable.

So this ruling was made, and I think the natural order recalculating patent was one of the first or might have been even the first.

Throughout the 80's, we didn't know about this. It was around 1990 that programmers in the U.S. started to become aware that they were faced with a danger from software patents. So I saw how the field worked before and how the field worked after. I saw no particular speed up in progress after 1990.

There was no political debate in the U.S., but in Europe there has been a big political debate. Several years ago there was a push to amend the Munich treaty that established the European Patent Office. It has a clause saying that software is not patentable. The push was to amend that to start allowing software patents. But the community took notice of this. It was actually free software developers and free software users who took the lead. [But] we are not the only ones threatened by software patents. All software developers are threatened by software patents, and even software users are threatened by software patents.

For instance, Paul Heckel, when Apple wasn't very scared of his threats, he threatened to start suing Apple's customers. Apple found that very scary. They figured they couldn't afford to have their customers being sued like that, even if they would ultimately win. So the users can get sued too, either as a way of attacking a developer or just as a way to squeeze money out of them on their own or to cause mayhem. All software developers and users are vulnerable.

But it was the free software community in Europe that took the lead in organising opposition. In fact, twice now the countries that govern the European Patent Office voted not to amend that treaty. Then the E.U. took a hand and the Directorates of the E.U were divided on the issue. The one whose job is to promote software is against software patents it seems. They were not in charge with this issue. It is the Open Market Directorate who is in charge, and is run by somebody who is in favor of software patents. They basically disregarded public opinion which has been expressed to them. They have proposed a directive to allow software patents. The French government has already said they are against it.

People who are working [on] various other governments in Europe to oppose software patents, and it is vital to start doing so here. According to Hartmut Pilch, who is one of the leaders in the European struggle against software patents, the main impetus comes from the U.K. Patent office. The UK patent office is simply biased in favour of software patents. It had a public consultation, and most of the responses were opposed to software patents. They then wrote a report saying people seem to be content with them, completely disregarding the answers. You see, the free software community said, "Please send the answers to them and to us too." So they published these answers which were generally opposed. You'd have never guessed that from the report that the UK patent office published.

They use a term that they call "technical effect". This is a term which can stretch tremendously. You are supposed to think it means a program idea would only be patentable if it only relates to specific physical acts. If that is the interpretation, it would mostly solve the problem. If the only software ideas which can be patented were those that really did relate to a particular technical, specific physical result that you might have patented if you didn't use a program, that would be OK. The problem is that you can stretch that term. You can describe the result you get by running any program as a physical result. How does this physical result different from any other? Well it is as a result of this computation. The result is that the UK patent office is proposing something that looks like it leads to mostly solving the problem and really gives carte blanche for patenting almost anything.

The people in the same ministry are also involved in the copyright issue, which really has nothing to do with software patents except that it is being handled by the same people. It is a question of interpreting the recent E.U copyright directive, a horrible law like the DMCA in the U.S., but there is some latitude for countries to decide how to implement it. The UK is proposing the most draconian possible way of implementing this directive. You could greatly reduce the harm it does by implementing it properly. The U.K. wants to maximise the tyrannical effect of this directive. It seems there is a certain group--The Department of Trade and Industry?--Who need to be reined in. It is necessary to put a check on their activities, stop their creating new forms of power.

Software patents tie up every software developer and every computer user in a new in a new form of bureaucracy. If the businesses that use computers realised how much trouble this can cause for them, they would be up in arms, and I am sure they can stop it. Business doesn't like being tied up in bureaucracy. Sometimes, of course, it serves an important purpose. There are some areas where we wish the UK government did a more careful job in tying certain businesses up in bureaucracy, like when it involves moving animals around. But in cases when it doesn't serve any purpose except to create artificial monopolies so that somebody can interfere with software development, squeeze money out of developers and users, then we should reject it. We need to make management aware of what software patents will do to them, get their support in fighting against software patents in Europe.

The battle is not over. It still can be won.