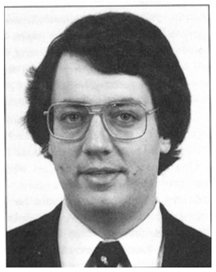

Freedom of Speech in Software

by Phil Salin 7/15/91

The following text is derived from the transcript of Phil's article as it appears in this message by John Gilmore. In this message, Gilmore says about this article

Phil Salin opposed software patents on free speech grounds, claiming a government monopoly over the use of certain ideas in software was exactly equivalent to censorship of literary ideas. This was the first application of the concept of freedom of speech to software.

To: Patent and Trademark Office, Department of Commerce
(E.R. Kazenske, Executive Assistant to the Commissioner,
U.S. Patent and Trademark Office, Box 15, Washington, DC 20231)

From: Phillip Salin, President, American Information Exchange Corporation
2345 Yale St., Palo Alto, Ca. 94306 (415) 856-1234

Re: Request for Comments, Advisory Commission on Patent Law
Reform (56 FR 22702-02)

Title: Freedom of Speech in Software

Date: July 14, 1991

The following comments relate to point I. "Protection of Computer Related Inventions", subpoint c) "The Supreme Court has found that new and useful computer program-related inventions are eligible for patent protection. What rationale, if any, exists in law or policy for Congress to now remove patent protection for this field of science and technology?"

SUMMARY OF POSITION.

1. Computer Programs are Writings. As such, they should be subject to copyright law (narrowly interpreted) or trade secret protection, but not patent law. As writings, programs should be protected against any attempt by government to license what can be written. This includes well-intentioned but mistaken legal or policy arguments which create de facto censors and censorship under another name, viz. patent examiners and patent examination.

Such censorship and restraint on freedom of expression of software writers is anathema in a free society, and a violation of the First Amendment. That software patents are a severe violation of the rights of speech of programmers has not yet been widely recognized; this is perhaps in part because most lawyers, judges and politicians are still insufficiently knowledgeable regarding computers to realize that writing a computer program is in fact a form of writing, not significantly more arcane than writing music, mathematics, scientific papers, or for that matter, laws. All of these forms of speech, including writing programs, deserve full protection under the First Amendment.

2. Central Planning or Licensure of Good Ideas in Software Won't Work. Just as any attempt to centralize or classify all original (or "non-obvious") literary, musical, or scientific writings in the patent office would fail, so any attempt to centralize information regarding all innovative software programs will also fail. No human can know all of software relevant to any large subject, just as no human can know all that has been written on any large subject, and for the same reasons. Current and near-term innovations in the writing of software will cause the amount of software developed every year by the one million professional programmers in the U.S. to grow at an ever-increasing rate. As a result, the burden of central licensing of innovation by the patent office will grow steadily more onerous, creating unnecessary and costly barriers to software progress.

3. Patents on Writings Discourage Trial and Error Perfection of Ideas. Rather than allowing government to restrict different expressions of the same important idea, by patents or otherwise, public policy should recognize that the more important the idea, the more important it is to support the freest possible variation of expression, in order to rapidly perfect the idea. The intense competition of commercial software in recent years, and the rapid improvements in software practice which the free market in software has hitherto engendered, strongly validates this theory in practice.

ELABORATION.

1. Computer Programs are Writings.

Anyone who has ever written both a program and an essay knows how similar these complex endeavors are. Both require use of all one's skill and knowledge. Both involve continual invention and creativity. Both require constant revision. Both evolve with time, as one's knowledge grows. Both are written in a language which has a vocabulary that can be used in an infinite variety of ways. Although software is often a less direct method of communication than prose, in that there may be many intermediaries between a particular programmer and the end-user of an application which uses a piece of his or her code, the same is true for other forms of expression. Theater goers, for example, don't directly read theater scripts, but see and hear them acted by intermediaries (actors); nonetheless, the scripts are writings.

Neither essays nor software are written with primary attention to "What has someone else said?", much less "What has the official licenser/patent examiner pronounced?" Rather, both are written with attention to solving a particular problem or achieving an objective of importance to the writer. In both cases, any similarity to the works of others normally comes about because of similarity in the problems which are being addressed, and not because of slavish copying of either ideas or implementations of others.

Writing programs today is no more esoteric than writing prose once was, and writing music still is. Until a few hundred years ago, literacy was a rarity. Acquiring the ability to write prose took training. It still does, but nowadays we teach writing to everyone in schools. Other forms of writing, such as writing music or writing computer programs, are treated as optional, but we still recognize them as writing. Even though the notes don't sing by themselves -- they have to be played -- we recognize the writing of music as a form of speech or expression.

Similarly, although a program has to be run to be used, before it can be run it has to be written. There are now millions of individuals in the U.S.A. who know how to write a computer program. It is an absurdity to expect those millions of individuals to perform patent searches or any other kind of search prior to the act of writing a program to solve a specific problem. If others wish to purchase a program, as with the sale of written prose and written music, absolutely no patent restrictions should be placed on the ability of authors to sell or publish their own writings. To do otherwise is to confuse the player piano (which is patentable) with the specific arrangement of notes on a specific player piano roll (which is indeed subject to copyright, but not to patenting).

Regrettably, the courts have allowed the patent office to be placed in the position of promoting de facto censorship of the work of over 1,000,000 employed writers of computer software, along with the several million additional amateur writers of computer software. All these millions of citizens are now asked to censor their own writings, or have them reviewed and censored by third parties or the courts. Whenever and wherever the patent office issues a software patent, software authors must now plead with patent holders to grant - for a price - licenses to speak as they choose; and the patent holder is under no obligation to grant these licenses.

Suppression of free thought and speech in software (writing, or publishing) is an evil, even when only a small number of individuals recognize that speech is being restricted, or what the costs will be if this harmful censorship-by-another-name, viz "patent licensure", is now allowed to expand unchecked.

The grant by any agency of government of the exclusive right to speak in software, and the enforced branding by government of all alternative expressions of the same or similar ideas (algorithms) as illegal is inherently harmful. It is to say, in effect, "Don't try to solve problems and invent solutions as you see fit; you or your software agents might independently write or invent something which the patent office's licensers have placed on the Index of Banned Algorithms; in which case, at their discretion, they can force you into an expensive, traumatic legal Inquisition..."

Under the First Amendment, the freedom to speak or write may not be abridged by any branch of government or by any government licensee. This holds regardless of the good or bad intentions of those who argue otherwise.

2. Central Planning or Licensing of Good Ideas in Software Won't Work.

Requiring writers of software to know all potentially relevant patents is the same as requiring writers of literature to have read all potentially significant works of literature before ever setting pen to paper. It is not possible; and even if it were, it is not desirable.

The main costs of software patents are not in the past or present; they are in the future. As programming literacy increases, and as Object-Oriented Programming, Genetic Algorithms, Neural Networks, and Computer Aided Software Engineering techniques expand, the volume, breadth, complexity, and scale of software written will continue to expand exponentially.

As this happens, the percentage of all that is worth knowing about software which can be known by a single person will continue to drop exponentially. This applies to software writers, and also to software readers, users, and reviewers, including patent examiners. No one can know much of what is non-obvious and innovative in software, even today; tomorrow, the problem will be even worse.

We are now entering the era when programs can write and edit other programs, and where it will simply not be possible for anyone to know which programs have evolved or been automatically revised to the point of similarity with other programs, innovative or not. Furthermore, new programs and algorithms will be written at the rate of the computer programs running to create them, not at the rate of the computer programmers typing code line by line.

At this point, software patents will go from being unworkable to being widely and deservedly recognized as impossible. For not only will they become impossible to enforce; they will become impossible to comply with. In the meantime, real companies will have to pay real lawyers increasing sums to try to avoid lawsuits, negotiate otherwise unnecessary cross- licensing agreements, and continually waste time, money, attention and energy on these and other defensive, rear-guard activities which will add nothing to America's productivity or actual stock of inventive software.

Allowing patents in software is tantamount to asking each individual programmer to know what all the other millions of programmers on Earth are currently doing or have already done. Requiring such omniscience by software writers is a sure way to force them into civil disobedience, if not intentionally, then after they find that they increasingly and unwittingly are nevertheless violating patents of which they cannot adequately remain current or aware. When the law begins to tell people to do what is impossible, disobedience and disrespect for the law are the inevitable result.

Thus, the only thing which software patents can do is create increasingly arbitrary and costly roadblocks to progress, including crucial progress in using new tools and techniques for writing software.

Since promoting innovation is the primary rationale for patents, and since patents instead serve to impede innovation when it comes to software, and will increasingly impede innovation as the volume of software continues to expand, it is quite clear that patents should not be applied to software writings for this reason alone, independent of the First Amendment issues discussed earlier.

Similar considerations argue against a broad interpretation of copyrights, including broad "look and feel" claims. The burden of proof should always be on the copyright holder to prove substantive and thorough copying or reverse engineering. Like writers of literature, writers of software should not be expected to constantly look over their shoulders out of fear that someone someday may try to sue them due to partial or accidental similarities with the works of others. Broad interpretations of copyright are even worse than patents in this respect, because claims of copyright infringement can be made by almost any party in any field, without significant time limit, and without any requirement to state publicly in advance exactly what aspects of one's work are sufficiently unique to deserve copyright protection. As a result, the idea of copyright protection, coherent when applied to a specific piece of writing, becomes incoherent and requires de facto omniscience if applied to all possible variations on all existing pieces of writing.

Because the focus of this analysis is on patent law and software, further discussion of the dangers of broadly interpreted copyrights should take place elsewhere.

3. Patents on Writings Discourage Trial and Error Perfection of Important Ideas.

Whenever a patent is granted on a particular expression of an idea in software, it will have a chilling effect on everyone who is considering writing software to solve similar problems. They must now tread gingerly lest their writings be later judged to overlap with the area that has had exclusive title granted to the patent holder; this holds even if their expression was independently derived, and even if it is a superior solution to the same problem. Given the manifest unpredictability of court decisions, many people will prudently decide not to innovate in areas in which someone has already been granted a patent. This is a perverse way to encourage invention.

Favoritism by a government official towards "non-obvious" (i.e. important) software innovations is like favoritism by government towards specific religious writings or specific commercial products in the marketplace: it's an inherently bad idea. The idea that one party's writings on a subject shall be certified by the Executive branch of the government, and all other expressions suppressed, would appear very strange to those who wrote the constitution.

Like a work of fiction, the value of a sophisticated work of software is not in the simple plot idea, but in the complex telling of the tale. It is only those unfamiliar with the strong feelings, beliefs and preferences which exist among writers of software regarding alternative expressions of the same software ideas who could believe that differences in expression of the "same" idea are unimportant to those who write software, or to those who use software written by others.

Imagine if, for 17 years, only one author was allowed to write about the plot line "boy meets girl, boy loses girl, boy regains girl". Or that once some consortium of artists has invented rock and roll or string quartets (and produced an initial "reduction to practice"), no one else could write music in those styles for 17 years without their permission. Or that once the first mathematician has invented a technique for dividing numbers, all other mathematicians must for 17 years request permission before inventing their own techniques, for fear of accidentally reinventing or coming too close to reinventing what another mathematician has also thought about. In each of these cases, imagine the arrogance of someone claiming a right to bring before a court of law and convict of a civil crime all others who choose to think for themselves and write independently.

Any assertion that some one individual or organization can ever rightfully establish exclusive ownership of the use or refinement of abstract ideas - obvious or non-obvious, important or unimportant - embodied in a work of prose, music, mathematics, or software, should trouble the conscience of everyone whose creative work is built, as it necessarily must be, in part or in whole, out of ideas and techniques discovered and developed by others.

The notion that the better the idea, the better it is to grant one author a monopoly over all possible expressions of that idea is perverse. It is precisely the most important ideas which deserve the most varied and thorough exploration. The more important or innovative the area of discourse, the more important it is to avoid government favoritism or censorship of thought or expression. Letting the government grant monopoly licenses to only the most important ("non-obvious") new ideas does not make such licensing better, but worse.

In addition to being harmful to the freedom of expression and experimentation with ideas by others, patents on software are also unnecessary. As with any complex novel or movie, significant works of software necessarily involve significant complexity and detail. Barring direct copying, which can and should be prevented by copyright law, the expertise and judgement involved in creating and continually improving any complex piece of software provides inherent protection against would- be competitors; those who invest in writing such complex software master the intricacies of the problem they are solving far better than any pure copycat competitor is likely to achieve. Asking for more protection than this, however, is asking too much. In all walks of life one makes investments which one would like to protect. But where granting protection for some involves imposing licensing requirements on others, we should always err on the side of forbearance. Software which is complex and original enough to deserve patent protection doesn't need it; and software which is simple enough to require patent protection doesn't deserve it.

Like censorship of religious speech in the 17th century, the issue here is not what the original justifications were (software patents as incentives to invention, etc.), but rather, how we can eliminate a dangerous, but avoidable, error: restrictive licensure of software; and how we can, instead, re-establish full freedom of speech in the writing and publishing of computer programs.

ACKNOWLEDGEMENTS.

These comments have benefited from review and suggestions and assistance of many colleagues, including: Ravi Pandya, Mark Miller, John Walker, Marc Stiegler, Chip Morningstar, Chris Hibbert, Chris Peterson, Eric Drexler, Eric Tribble, Nick Whyte, Gayle Pergamit, Roger Gregory, Robin Hanson, Michael McClary, Paul Baclaski, Rick Mascitti, Kimball Collins, and Bob Schumaker. Any error or inelegance of expression, however, remains the responsibility of the author.

The ideas in these comments have been evolving over a period of years, and may be to some extent original with the author. However, there is a wider intellectual climate within which they are shared. Several colleagues who share the general views expressed in this document have attached their signatures. They are almost all writers of software.

Authors of non-software whose writings have heavily influenced my thinking regarding the severe dangers and foolishness of even attempting to centrally plan or license spontaneous areas of human endeavor such as the writing of software, are: F.A. Hayek (Law, Legislation, and Liberty); Karl Popper (Conjectures and Refutations); and Michael Polanyi (Personal Knowledge).

As President of a small company which is currently working day and night on developing an innovative software-based product, I do not have much free time to participate in public policy issues. I had not until recently planned to attempt to write my views on the subject of software patents and copyrights until a few years from now, when I would have more leisure to fully research, critique, and elaborate on them. However I have decided to make these views public at this time, in somewhat "raw" form, as a contribution to regulatory proceedings which I consider to be of the utmost importance.

I apologize for any lack of familiarity with the fine points of current patent law as it is currently being applied to software. My whole point is that any such details are necessarily irrelevant once one recognizes that patents should not be applied to any form of writing, including the writing of computer software.

---Phil Salin, July 14, 1991.

We the undersigned are in substantial agreement with the argument here, and hold that for these reasons as well as others, patents should not be granted in computer software.

Eric Drexler
President
Foresight Institute

Roger Gregory
Chief Scientist
Xanadu Operating Company

Robin Hanson
Artificial Intelligence Researcher
NASA Ames Research Center

Chris Hibbert
Manager of Software Development
Xanadu Operating Company

Richard J. Mascitti
Manager, Hypermedia Software Development
Autodesk, Inc.

Michael McClary
Software Quality Control Tools
Xanadu Operating Company

Mark S. Miller
Co-Architect
Xanadu Operating Company
Co-Director
The Agorics Project
George Mason University

Chip Morningstar
Vice-President of Software Development
American Information Exchange

Ravi Pandya
Co-Architect
Xanadu Operating Company

Gayle Pergamit
Manager
American Information Exchange

Chris Peterson
Director
Foresight Institute

Bob Schumaker
Macintosh Programmer
American Information Exchange

Eric Dean Tribble
Co-Architect
Xanadu Operating Company

Comments?
email the webmaster

or Crit Me Now!