From: Jay Sulzberger [e-mail redacted]
Sent: Monday, September 27, 2010 11:53 PM
To: Bilski_Guidance
Cc: [e-mail redacted]
Subject: On Software Patents

Dear USPTO,

I write to express opposition to the granting of patents on software.

The arguments against software being patentable are many. I repeat here two arguments. The first argument has rubric and slogan "Software is mathematics and thus cannot be patented.".

This argument is powerfully persuasive to many programmers and mathematicians. Indeed most in these trades, when they hear or discover for themselves the slogan, consider the slogan form of the argument sufficient, and their opinion of the issue is settled at that moment. I wish to call attention to the great force of the argument and its tremendous and growing field of application.

Modern digital computers were discovered at least twice, both times by mathematicians. First, Charles Babbage in 1837 proposed a design for a digital computer, which computer was intended to calculate anything that could be calculated by any algorithm whatsoever. Second, a body of logicians and mathematicians of the period 1850-1940 engaged in a large attempt at understanding mathematical infinity and the meanings of mathematical truth and mathematical proof. Near the end of this period several different designs for a digital computer were published, with arguments that such computers really could perform any task for which an algorithm is available. Mathematicians, logicians, probabilists, and engineers together then, from 1940 to 1950, built such general purpose computers. The general purpose computer is a very astonishing machine. It is different in kind from any machine ever built before by human beings. Any general purpose computer can do what any special purpose calculating device can do, when the general purpose computer is fed the proper program. For this reason logicians call a general purpose computer a "universal machine".

Now the Patent Office considers that no "general concept" can be patented. Today an ordinary digital computer, such as the one into whose memory I am typing these words, is an instantiation of the general concept "universal machine". Though various material components, and many of the processes which produced them, are patented, yet there is no patent on the general concept of a universal machine. And this is right. Such a patent would give to the patent holder a legal monopoly on all ordinary computers, and just about all devices for electronic communication, such as telephones, radios, televisions, etc..

Once one has a material instantiation of the universal machine, that is, a computer, then one has a method for taking any algorithm and running the algorithm on the computer. Or, more precisely, once you have written a program, in a specific computer language, then any compiler/interpreter running on any hardware is sufficient to perform the steps of the algorithm. No specific compiler, no specific interpreter, no specific hardware is required. The only specification is that they work together so as to run the program. Starting on page 3 of the pdf file

http://www.uspto.gov/patents/law/exam/bilski_guidance_27jul2010.pdf

is a section called "101 Method Eligibility Quick Reference Sheet". In the subsection called "Factors Weighing against Eligibility" we have this factor against eligibility to be
patented:

Insufficient recitation of a machine or transformation.

Machine is generically recited such that it covers any machine capable of performing the claimed step(s)

Every "computer program" meets this factor against eligibility.
The point of computer programming systems, such as compilers and interpreters, is that they mediate between particular fixed expressions of algorithms, that is to say programs, and one or more particular instantiations of the "universal machine". There is no particular requirement on the machine that is to run the program beyond the generic requirement that the software/hardware actually be able to run the program. We are squarely, deeply, in the center of this factor against eligibility.

Thus today, if the Patent Office follows its own guidelines, no computer program is patentable.

The second argument against granting patents on software is made by several other commentators. This argument would stand even if the Patent Office were to drop the "insufficient recitation of a machine or transformation" factor against patent eligibility.

As written today much software is a work of bricolage. Bits of programs and whole programming systems are welded together to make a program. Often most of these bits and systems are written by people other than those who produce the program. For example, Google has several large systems, which might be considered as single programs. The majority of the code in these programs was not written by Google. Rather this code by other authors was assembled by Google and made part of the Google systems. Much of this code

was published under a free license, such as the GNU General Public License, the Revised BSD License, the Apache License. This code is copyrighted, and many who are against software patents are happy to copyright the code they write and make it available under a free license. Such code is called "free software".

Because of this extraordinary degree of bricolage and assembly of code written by others, the granting of software patents would allow rich patent holders to suppress much use of free software.
The mechanism of suppression would work as follows: because most large free software systems, such as the GNU/Linux operating system, are composed of thousands of pieces of code written by hundreds of people, the user could not be sure that every piece was non-infringing. Thus businesses and other organizations would be dissuaded from using the GNU/Linux OS.

But even if software is made available under a restrictive license, even if it is a trade secret, yet almost every body of code would infringe on some software patents if software patents were granted. (Support for this statement requires an analysis of the differences between development of ordinary machinery/processes/materials and the development of software; please accept my apology for not including such analysis in this
note.) Thus software patents would damage not only authors and users of free software, but also authors and users of any software, because a patent suit can always be brought by a rich patent holder, and defense is expensive. Thus software patents tend toward creating an oligopoly of large companies with the power to prevent the production and use of software written outside the cartel. This danger is not notional. Microsoft has claimed that authors of competing OSes are in violation of some Microsoft patents, and Microsoft has succeeded in extracting license fees for use of OSes which Microsoft did not write a single line of.

I remain your fellow user of free software, such as the software on which the Net runs, and fellow student of history and probability, Jay Sulzberger.