By Paul Belleflamme, 3 October 2013

Fixing the software patent problem

In an article published the New York Times on October 7, 2012, Charles Duhigg and Steve Lohr give a critical view of the current state of the US patent system:

> *Patents are vitally important to protecting intellectual property. Plenty of creativity occurs within the technology industry, and without patents, executives say they could never justify spending fortunes on new products. And academics say that some aspects of the patent system, like protections for pharmaceuticals, often function smoothly.*
>
> *However, many people argue that the nation's patent rules, intended for a mechanical world, are inadequate in today's digital marketplace. Unlike patents for new drug formulas, patents on software often effectively grant ownership of concepts, rather than tangible creations. Today, the patent office routinely approves patents that describe vague algorithms or business methods, like a software system for calculating online prices, without patent examiners demanding specifics about how those calculations occur or how the software operates.*
>
> *As a result, some patents are so broad that they allow patent holders to claim sweeping ownership of seemingly unrelated products built by others. Often, companies are sued for violating patents they never knew existed or never dreamed might apply to their creations, at a cost shouldered by consumers in the form of higher prices and fewer choices.*

US vs. EU

The extent to which software should be patented has been – and still is – a topic of intense debate. Nowadays, there is no harmonization across countries on this matter. As indicated in the quote above, the US Patent and Trademark Office broadly grants patents that may be referred to as software patents (and it has been doing so since at least the early 1970s). In contrast, the

European Patent Office has adopted a more restrictive approach (see here):

> *The truth of the matter is this: Inventions that use computer programs to provide a business process – not a technical process – are not patentable.*
>
> *A patent application for an Internet auction system was not granted because the system used conventional computer technology and computer networks – which meant it made no inventive technical contribution to the level of existing technology. Such a system may provide business advancement to its users, but that is not the type of advancement required by the EPO.*
>
> *On the flip side, the problem of improving signal strengths between mobile phones is a technical problem, even if it is solved by modifications to the phone software rather than its hardware. Such an invention would obtain a patent, provided that the solution is also novel and inventive.*
>
> *In this respect, the granting practice of the EPO differs significantly from that of the United States Patent and Trademark Office (USPTO), where patent protection for software is granted, even if it does not solve a technical problem.*

Adjusting patent subject matter, length, and/or breadth?

The contrasting approaches of the USPTO and the EPO are in terms of "patentability" (or *patent-eligible subject matter*): what are the conditions for a software (or a "computer-implemented invention" in the European terminology) to be patentable? Two other, and potentially complementary, ways to tackle the issue would be to discuss the "patent length" and the "patent breadth" for software.

In terms of *length*, some scholars, among them Richard Posner (Federal appeals court judge and professor at at the University of Chicago Law School), recommend a shorter patent term for digital technologies. How long should software patents last? Although Posner does not propose an explicit length, others do. For instance, according to the Electronic Frontier Foundation,

> *A patent covering software should survive for a term of five years, beginning from the date the application is filed.*

Other scholars suggest to reconsider the *breadth* of software patents. For instance, Mark Lemley suggests to limit software patents to their specific way of accomplishing a function. In his paper "Software Patents and the Return of Functional Claiming", he claims that the real problem with software patents (in the US) is the following:

> *Software patent lawyers are increasingly writing patent claims in broad functional terms. Put another way, patentees claim to own not a particular machine, or even a*

> *particular series of steps for achieving a goal, but the goal itself. The resulting overbroad patents overlap and create patent thickets.*

As summarized by Eric Goldman in a Forbes article:

> *Prof. Lemley argued that many software patents use "functional claiming," which is patenting a software function (basically, the problem that needs to be solved) rather than a specific way to implement that function (the innovator's solution to the problem). For example, currently we allow patent claims in the form "a computer programmed to achieve this result" or "a computer programmable/capable of achieve a result" (his research identified 11,000 patents using the "capable of" language). Prof. Lemley argued that we should prevent functional claiming in software, allowing patents only on methods of achieving the function, not the function itself.*

And you? What do you think? If you had to weigh the pros and cons of software patents, what patent duration and breadth (if any) would you recommend? To help you organize your thoughts, you can use the framework proposed by William Nordhaus in his book "*Innovation, Growth, and Welfare*" published in 1969 (see Belleflamme and Peitz, 2010, pp. 517-521 for a textbook presentation).

(This is an edited version of a post that I published on this blog in October 2012.)