*11 questions are stated about the definition and impact of the perimeter of patentability in relation to algorithms, software, data structures and information processing methods. These questions regard the overall legal framework, the substance of what software is, the economical context in which software innovation occurs, the relations between hardware and software, the sociology and financing of innovation, and the institutional context. The main objective is to set the basis for further research and to provide some preliminary directions for the impact assessment of the scope of patentability on research, innovation, competition, literacy and societal trust in technology.**

11. *Can the difficulty of software patent examination be handled with more and better qualified examiners supported by better tools?*

1. *What is "in a field of technology"? What does TRIPS command to be patentable?*

Several legal analysts have interpreted TRIPS article 27.1 as commanding software to be patentable. Even if TRIPS did command such a thing this would be no reason to stop thinking about whether it is a good idea. Commentators as distinguished as Joseph E. Stiglitz have recently stated that “almost every aspect of this agreement must be reconsidered”[1]. Such a radical review is probably needed for many reasons. But this does not mean than it should today be accepted in any way that TRIPS commands software to be patentable. TRIPS article 27.1 states that “patent protection shall be available for inventions, whether products or processes, in all fields of technology”. The 2 essential qualifications “whether product or processes” and “in all fields of technology” have been overlooked by legalistic commentators, because, in their reading they were mostly intended to be inclusive, to make it impossible to exclude or treat with particular conditions a field of technology. Similarly, for some industrial property thinkers, there is hardly a thing in the world which is neither a product nor a process.

But is software in a field of technology? Is it a product, as the European Patent Office has held when developing its recent case law about software itself as a product? It is always a particular pleasure for someone working as “head of sector for software technologies” in a programme called “Information Society Technologies” (forbid the plurals), to argue that certainly, software is not in a field of technology. Software is the encoding of a text in a formal language, which can under certain conditions be translated into other encodings of other formal languages and can then, in some particular execution environments, be executed by computer(s), and eventually produce some effects in the computer considered as an abstract machine, such as changing the value of some memory. These values can be mapped by some hardware into presentations of information or parameters of controls of physical devices. Let us consider the following program:

```
main(){

    printf(“Hello world!\n”);

}
```

Is this program a piece of technology? Obviously not. It is a text printed on the page or screen you are reading. But it “represents” a piece of technology will claim the legalistic analysts. Drawing of machines represent pieces of technology, and one does not allow patents on the drawing. But one allow patents on the underlying ideas on an invention that drawings in patents are only a way to convey to the reader, just like here the software text conveys the idea of the invention residing in its execution, will the legalistic analysts claim. But does it? Will this program compile to a syntax error? Certainly if you feed it to a Prolog compiler. Will it, if compiled by a C compiler, produce a run-time error? Most probably if it runs on the wrong processor. Will it, when running, erase your hard disk? Could be if it is executed with the proper authorisation rights and with its output piped to the proper file. Will it make the Ariane rocket explode? Who knows, if it is loaded as a component of a system and other components use its output to map it to their acceleration input parameter.

Far from being hair-splitting arguments, these remarks highlight the fact that not only is software in itself intrinsically non-technical, but it takes a good lot of precision to understand how it can be mapped into some field of technology. Don Knuth beautifully summarised all this in one warning: “Beware of bugs in the above program, I have only proved it correct, not tried it”[2]. Software is no more a product than it is a piece of technology. It can be sold as a product, though in most cases it is licensed, and thus sold as a service (even if you buy the license). It can be packaged on a product. So can novels. Do we patent ideas of novels?

At this stage, my fictitious legalistic analysts must have reached indignation. Doesn't everyone know that software is everywhere in technology of all kinds, that it accounts for large shares of expenses in many industries, that software even replaces hardware. The next questions propose to give a closer look at these affirmations. Meanwhile, we can remember that we need precise criteria for knowing under which conditions software can be mapped in technology, and that software itself does not seem to be one. One last remark on this: contrarily to most texts, software is often produced by software. Even patent infringing software can be automatically generated by non-patent infringing software[3]. It is only human that it can be difficult for some to resist the prospect of such an infinite ocean of litigation.

2. *What are the typical types of software patents?*

   From an extensive reading of software patents, I came to the conclusion that there are 3 main types that account for the very vast majority:

- Patents on features or elements of functionality: typical examples are patents on “drag and drop of a URL text to create a shortcut on a desktop” or on “multiple granularity interactive selection in text based on segmentation of text into words”
- Patents on components of software systems: A typical example is “caching and access management based on a markup language used for browsing the Web from wireless devices”.
- Algorithmic patents: a typical example is “detecting scene transition in video by statistical analysis of motion vectors”.

It would be interesting to conduct an in-depth analysis of the distribution of patents according to this typology. Unfortunately, this is a time-consuming research. The author has applied it to limited sets of patents (the 30 most recent patents whose text includes the word software granted to a particular company), and this represented about a week of work for someone with a good general software and information technology culture.

The main difficulty is to distinguish patents on components of software systems (that are software patents in my definition) from patents on physical processes or devices based on the execution of some software components. The main criterion I have used is to look at the claimed technical effects, and to see if these effects are themselves physical (that is the benefit resides in how the forces of nature and organisation of matter are used) or informational (separable from their physical basis). When using such a criterion a video memory caching software technique for quicker refresh of windows in a graphical user interface is informational, while a software optimisation of low energy consumption in a processor may be physical. But this distinction, clearly stated in the case law until the mid-eighties, has been intentionally blurred in further evolution. As a result many patents are written today as software system component patents (to cover the wider possible scope) while a more limited version of the patent might also have been written to protect the same “invention” as a physical process.

In my experiment, conducted for a company that has a significant hardware as well as software activity, the results were that around 70% of the patents were pure software patents, but almost half of these could potentially have a more restricted version which would have held under a stricter patentability scope.

Note that this discussion concerns only scope of patentability, independently of the judgement one has on the novelty, originality or inventiveness.

3. *What is the size and granularity of investments in software innovation? At which stage of the software life cycle do they occur?*

   One has to reconcile 2 apparently contradictory facts. For one thing, organisations, both companies specialised in information and communication technology and organisations using it, spend huge amounts of money on software. For another thing a given piece of independently marketable software rarely accounts for more than tens of person-years. If you look at an invention as described in patent texts, the corresponding development is very rarely more than a few person-years, and in most cases not more than a few days. We thus have a peculiar situation in which software systems are expensive to build, more expensive to test, even more to deploy and evolve, but independent innovation components in them are small and most often incremental. Of course it can take a good deal of skills that are expensive to train, hire and keep to build the right software component, or to choose the right algorithm to apply to a particular universe of data. To summarise it in one sentence: *software is small bits of innovation, on the shoulders of a large infrastructure of skills, tools and pre-existing software, that are complex and expensive to deploy, test and evolve into real world applications*. When software has reached a sufficient level of usefulness, it can be spread extremely quickly in markets. All other things being equal, the first mover advantage, and the increasing returns that it brings compensate largely for the - relative - ease in cloning a (proprietary) software system. From all this, it results that there are not any risks that the absence of patent protection would create a disincentive for software innovation in the context of software systems. Economic studies and models[4] and empirical evidence such as the huge investment in software innovation before patent protection became available are convincing in that respect. It would nonetheless be interesting to have more detailed analysis of the patterns of investment depending of type of software systems and markets.

   When presented with the evidence above, the supporters of software patents in Europe recourse to the embedded software argument. When a given product or process rests on a combination of software and hardware, as is the case for embedded systems or systems-on-a-chip, the investment pattern and the product deployment economics are a combination of 2 very different types or life cycles. They claim that by not providing patent protection to the software innovation components of these mixed systems, one could undermine investment in the design and manufacturing of these systems. The issues related to this special case are discussed below in the next question. Meanwhile, let us remember that any patent protection that would be granted should be defined in such a way that it does not extend to the cases where it is obviously irrelevant.

4. *Is hardware being replaced by software? Which protection for hardware / software co-design inventions?*

   Is hardware ever replaced by software? Obviously not. No software ever ran without a hardware. But it is frequent that a hardware system is replaced by the combination of some new form of hardware (for instance embedded computers, sensors, and effectors) and software. What people mean when they say that hardware is replaced by software is even more general: they see that software and generic information technology accounts for a bigger and bigger share of the value chain in many industries. More generally, let's examine what happens when a classical technology is reshaped by the introduction of software components, or by the introduction of layers of information entities (such as gene sequences)[5]. The software and information entities themselves have properties that do not justify or strongly call against granting patent monopolies, but a physical invention using them often still meets the criteria for patentability. This tension has lead to an intense pressure for an increased patentability of software or information entities themselves, which has progressively introduced itself in case law and in the practice of patent offices. A number of dangerous mistakes have developed, such as the idea that a gene sequence "would represent a biological function, and its usage in a possible drug, treatment of biotechnological process" or that a computer program "would represent its execution on a generic computer, and the usage of this execution in a technical process". These statements are factually wrong[6], and they have been used to justify the unjustifiable: patenting discoveries, human expression and ideas, when none of the usual criteria for patentability was met. One important aspect of this discussion is that one must assess what patent protection will become for technical devices and processes when larger and larger chunks of them will be accomplished under the control of software or information processes. As a thought experiment, let's imagine that a set of formerly mechanical processes would become one single meta-mechanical machine-tool, where a software would control which of the former individual processes is accomplished by the meta-machine. The meta-machine itself would deserve patent protection for its technical implementation innovative aspects. But each of the individual instantiating software should not be covered by patents. To understand why, one can consider 2 aspects of this change. The first one is that the motivation to introduce it is naturally to replace a complex and rigid mechanical design process by a less costly and flexible to rearrange software development or information process. It is exactly because the software part of the new process no longer meets the criteria for patentability that investors might want to introduce it. But in addition, contrarily to what classical patent thinkers always assume, the software components, considered themselves, are not mechanical inventions. They are nothing but algorithms mapped though the particular semantics of an execution context into the realm of mechanical processes.

It would be useful to complement this analysis by obtaining better empirical knowledge of the patterns of investment and technology life cycles in mixed hardware / software innovation.

5. *What is the impact of increased patentability of algorithms on research and innovation?*

   Algorithmic patents are often produced by academic researchers. The technology transfer departments of large public or semi-public research labs in information technology (CNRS, Max Planck Institute, Fraunhofer-GMD, INRIA, etc) often advise or even try to require, possibly with the assistance of law and regulation, for their researchers to file algorithmic patents. This is a truly paradoxical situation since there is good evidence that algorithmic patents are particularly detrimental to the progress of academic research. The analysis that follows is not substantiated here by any hard evidence. It is based on my limited experience and vision. It would deserve to be confirmed or contradicted by hard facts.

   Algorithmic patents hold (at least in the view of intellectual property division managers) the promise of the holy grail: licensing revenues to fund more academic research. They even start to pay off in academic career plans, which open the scary scenario that we could witness as explosive a growth of their number than for technical papers. Software system component patents might also look promising[7] for those organisations playing the spin-off game, but algorithmic patents seem to reward directly academic work. Unfortunately, algorithmic patents seem to have a major side effect: they slow down cumulative progress in computer science. In my experience, this slowdown occurs through 3 channels. The first one is the very weak disclosure effect of patents, due to the secrecy period and secondarily to the lack of clarity and validation data in patents. This has a worse effect in Europe than in the US due to the absence of grace period: researchers have to abstain from publishing while their technology transfer departments assess the patentability and possible value of their work. But even when a patented algorithm is known, it keeps slowing down algorithmic innovation. This occurs through the second process: channelling resources towards looking for a non-patented technique for achieving a result that was already obtained. This is a typical example of the essential difference between physical inventions and information inventions: an algorithmic patent is either irrelevant (if there are 1000 other algorithms to achieve the same result) or impossible to turn around (if it captures an essential mathematical constraint of the problem space). In contrast, for physical (f. i. mechanical or chemical process) inventions, there are often a limited but open number of solutions that improve on each other by removing constraints associated with the previous one. Thus turning around a physical invention patent is a source of progress, while turning around an information invention patent is either lost time or impossible. Finally, the third process is that, since patented algorithms very seldom have free or open source software reference implementations[8], it is extremely difficult to conduct scientific comparisons of efficiency of algorithms. The rare works to that effect often re-implement with errors or missing elements the algorithms under comparison. These very few published comparisons are of little value in such a situation, and there is a resulting lack of scientific knowledge about how algorithms perform. In comparison, fields characterised by the existence of freely available software implementing non-patented algorithms demonstrate a much quicker progress. It is illustrative to compare fields that have reached maturity before the algorithmic patent fad (for instance general algorithms from Donald Knuth's “The Art of Computer Programming” or basic image processing and digital audio processing) with fields like video scene change detection (a relatively easy problem with many patents, and still no widely available solution 10 years after several research teams have solved it to a very satisfying degree).

Seen from the public research funding angle, algorithmic patents lead to a pattern of repeated funding of projects redoing various flavours of the same thing. Even worse, fields with algorithmic patents are characterised by a lack of usage and exploitation of the results of these projects: while in principle we should witness the dream scenario of licensing, what we actually witness is innovation buried in patent portfolios, and not exploited at all. The reason is simple: the true patterns of diffusion of software innovation are so unpredictable that only wide and transaction free availability actually spreads it. Patents and licensing for academic organisations are an overall losing game with a few winners, the worse possible situation, since elementary psychology and institutional sociology tells us that the losers will keep asking for more resources and keep gambling as long as they can. Even worse, the technology transfer divisions have such an interest in hiding the truth, that only a forced disclosure of costs and revenues associated with patenting and licensing is likely to help us in reaching a better knowledge. Further research must not consider patent intellectual property in isolation: it must account also for the lost time for research, and the lost opportunities for others forms of dissemination.

6. *How will software and information process patents impact user literacy, inter-operability and competition?*

   Whatever decisions are made on the scope of patentability, the impact of software-related patents is largely to come in the future. From my experience, one of the main sources of misunderstanding around these issues is that some consider software a field evolving towards a classical engineering maturity, while other see software and information handling as a general purpose literacy, as important to tomorrow's society and economy than reading and writing is to today's. It is obvious that both views are grounded in some reality. For some form of programming and information handling to become pervasive literacy, it will have to look and feel very different from what we know today, and to that purpose one will need advances in software technology and engineering of complex systems made of autonomous parts. But it remains that we need some caution on the potential impact of treating with industrial property instruments the basis for future literacy, and for all societal and economic activities. The consequences of easing the creation or maintenance of monopolies and oligopolies would be all the more important since there are already natural trends to such concentration for information-related activities characterised by increasing returns.

   It took 5000 years for literacy in the classical sense to spread out of limited groups of scribes, monks and scholars. For recorded time-based media (more than a century old), though camcorders and sound recorders are readily available, the concentrated structure of media industries, their hard fight to protect their business models and access to attention time have up to now successfully kept media literacy underground, even if it seems that harder and harder means have to employed to that effect. Before patentability spread, information and software technology fought for people's interest by demonstrating that they could empower them. The key successes of information technology literacy have been built upon openness (email, Web navigation and publishing) or under the instruments of copyright (word processing). Now that some major dominant positions are installed and that other business feel threatened in their business models, there is a genuine risk of trying to locking out further progress of literacy.

The key issues in that respect regard consumer and citizen trust in technology, and entry / complexity costs for access to provider roles. The next phase is characterised by strategies of some players based on Web services, net identity management, user data management, and rental business models. One can have major fears that software method / process patents will be used to install major competition locks in this phase. Impact analysis might come too late, when irreversible harm is done. One has to consider carefully if we can afford taking such as risk.

7. *What credits do software innovators need for getting access to financing?*

   One of the arguments for software patents that has some ground for consideration regards access to financing for software innovators. It is well known that venture capitalists and technology transfer support programmes feel reinsured by patent credentials. But do the statistics of risk investment support that impression? In 1999, cumulated granted software patents in Europe were 10 times less frequent in proportion of total patents than in the US. The overall technology venture capital investment was close to 4 times bigger in the US, but the share going to software was greater in Europe (30% to 48%[9], 34 in the US)[10]. Independently of this debate, it remains certain that better credits for software innovators would be useful. Not only patents, but also publications poorly account for contributions to software innovation. The free / open source software world has developed informal mechanisms of acknowledgement of contributions. Even if these mechanisms are not primarily oriented towards rewarding innovation, they show the way of using peer esteem mechanisms to build credits, and it would be worth generalising them and giving them some technical support and visibility.

8. *What is the impact of accepting or rejecting software and data structure / format claims?*

   Software and data structure claims have an object of particular discussion in Europe because of the impact the form of claims has on the definition of infringement. With software claims, any author, copier, distributor, publisher of a piece software based on the same underlying principles than the patented invention is in direct infringement. Without software claims, only contributory infringement can be charged, which lowers the prospects of actual litigation against independent developers, Web site publishers and Internet Service Providers. This explains that even if the scope of patentability is not necessarily different, the practical impact of patents is different, in particular for free / open source software.

   One should nonetheless note that, even without software claims, the same effects of barriers to market entry and competition, litigation threats, and obstacles to interoperability can be obtained with any combination of:

- a weak definition of technical character in patent law,
- uncontrolled extensions to copyright concerning protection technology and outlawing of circumvention (DMCA, and the European Directive on Copyright in the Information Society depending on its transposition in National rights),
- licensing regulation that allow to contract around inter-operability provisions (UCITA).

9. *Who controls and what drives the European Patent Office?*

   In a feat of statistical presentation the patent offices proudly announce that 70 to 72% of holders of patents granted in 2000 own only one of these. The other side of the coin is that 7% of the patentees (whether in Europe or in the US) hold half of the total granted patents in 2000 (source: Trilateral statistical report, www.european-patent-office.org/tws/tsr_2000/ with some computation from graphs 4.3.1 and 4.3.3 by the author). For the US (source USPTO), the 50 biggest patentees for 2000 were granted as much as 24% of the patents for that year. 40 of these 50 companies are software, information technology or media technology companies to a large extent.

   When one considers the owners of patents in force, the concentration is much more important. It is difficult to estimate exactly from published figures, but figures computed through rigourous statistical sampling of patent databases by FFII (http://swpat.ffii.org/vreji/pikta/perled/app_stat.html) show that 50 companies hold 43% of 38852 EPO software-related patents, and the top-10 companies hold 24%. Now where does the EPO (like most patent offices) gets its money from? Not at all from public funding. A predominant part of its resources are generated by maintenance fees on patents in force, and most of the rest from application fees. The European Patent Office is an incredibly profitable organisation. Its operating surplus in 2000 is 284 million DEM, 20% of its operating income. This is almost Microsoft-level performance, and allows the EPO to have a staff growth of 10% per year, no doubt opening the way to further growth.

   Is is thus clear that the EPO (as other patent offices) has a strong dependency on its major customers. The effects of this dependency are not just some ill-intended generalisation. IGEPA/SUEPO/USOEB, the unions of EPO staff wrote in 2000: “Revenues of the self-financed EPO being directly proportional to the number of granted patents, its direction implicitly encourages examiners to do a slapdash work in order to generate more income. Examiners are thus in front of a very unpleasant choice between conscientiousness and the career opportunities that were set in front of them at recruitment time”[11]. One has to account for possible union bias, but figures regarding acceptation rates are also indicative: acceptation rates went slightly down in 2000 from the even higher rates of the previous years, but there were still 27523 European patents granted that year (all patent domains), for 45764 examinations[12]! Some will say this does not matter since EPO is a public organisation, with a treaty convention as legal charter, and it is controlled by an administrative council where governments are represented. Unfortunately, there is more evidence of patent offices controlling government and public administrations than of the reverse. 16 of 21 members of the EPO's administrative council[13] are … executive officers of National patent and intellectual property offices, themselves controlled by departments of ministries predominantly staffed by … former or present employees of patent and intellectual property offices.

10. *What is the real difference between USPTO and the most recent EPO practice?*

    The evolution of the scope of software patentability has been progressive, and this evolution started much earlier in the US. It accelerated significantly from 1997 at EPO. In the early 90s, there was a major difference between USPTO practice and EPO's practice. But what about now? The legal framework is still very different since the European Patent Convention excludes in its article 52(2) computer programs, presentations of information, methods for doing business, mental acts, mathematical methods and other entities from being inventions. But this exclusion and the limits to it stated in art. 52(3) have now been interpreted to the point of being totally ignored. In his statement after the after representatives of governments in the Munich Diplomatic Conference decided to postpone revision of article 52(2), Dr. Roland Grossenbacher declared on 29 November 2000: “Technical solutions for use in data processing of for carrying out methods of doing business therefore remain patentable”[14]. EPO went a step further when it published in October 2001 revised examination guidelines[15] authorising software and data structure claims, as indicated by some case law decisions from its in-house Chamber of Appeal. As of now, there remain some microscopic differences in scope of patentability, mostly linked to the difference between “usefulness” in the US patentability framework and “susceptibility of industrial application” in the European one. But the practical implementation of these differences does not give rise to any real disagreement on technicity: chapter IV.9 of the revised guidelines now makes it a real challenge for something NOT to be technical. There is a 3-4 years inherent delay before such evolution can be seen in granted patents (18 months for published applications).

The European Commission has announced that it will publish before the end of 2001 a directive on the patentability of “computer-implemented inventions”. Its exact contents are not known. The text submitted for public consultation in October 2000[16] did not propose to accept software claims but apart from that simply codified the practice of EPO. Recent press articles[17] have announced that software claims will not be accepted in the final directive proposal.

11. *Can the difficulty of software patent examination be handled with more and better qualified examiners supported by better tools?*

    USPTO itself, and authors such as Greg Arahonian have argued that the triviality of software patents is mostly due to insufficient resources, qualifications and prior art access tools in patent offices[18]. With all respect to G. Arahonian, whose work was essential to bring the software patent crisis to public attention, I dare to disagree. I claim that finding software prior art is an essentially impossible task, because it is an ill-defined problem and will remain so. Let me take an example. US patent 5,574,840 and its continuation 5,832,528 by Kwatinetz, et al. (Microsoft) describe variable granularity selection in text, that is the feature implemented in MS-Office of changing the selection granularity from letter to word when passing a word boundary, and various mechanisms for interactive control of that feature. US 5,574,840 is by far not the worse software patent that I have seen. US 5,832,528 is an extension using disguised software claims (18, 20, 31 and 33 start by *“a computer-readable medium whose contents cause a computer system to ...”*).

    My students and myself implemented a more general version of variable granularity selection for both music and video from 1993 to 1995 in software that was used by users in libraries and archives. We did several communications and publications on this subject within a general interaction framework called discrete multi-scale representation-based interfaces, that never mentions granularity. Variable granularity selection is more complex for audio and video than for text since one has to automatically recognise meaningful underlying units that are obvious for text. Its usefulness is also more evident in the case of audio. When we were discussing it for audio, we would always use the text example as a kind of toy example. I am not claiming this constitutes prior art invalidating the Kwatinetz patents (it would take me a long time and possibly is impossible for me to check if we actually published or distributed software using the very idea of variable-granularity selection in a context that implies its application to text similar to the patent's and before it was filed). I do claim that this is instructive with regards to how software ideas are transported from one domain to another, and about how software prior art may be hidden in the most obscure places, under improbable names.

When pushed to the limit of recognising the obvious impossibility of prior art search for software ideas, those who think the problem is fixable use the argument of opposition procedures. But the difficulty of finding related prior art also applies to finding which patents you should oppose. I discovered the Kwatinetz patent by chance, when looking for an example to illustrate the difficulty of interaction design choices, and more specifically how a useful functionality turns to be a nuisance when the user wants to do a thing that the designer did not forecast.

The very beauty of software and algorithms is that through from repeated steps of modelling, abstraction, semantical mapping, composition and metaphors together with lots of hard tuning, innovation and usefulness can arise. It is a beauty that all can contemplate, to which many can contribute, but that dries out when you try to hold it for yourself only.

About the author

I am Head of Sector "Software Technologies" in the unit "Technologies and Engineering for Software, Systems and Services" of the European Commission Information Society Technologies R&D Programme, in which I am in charge of actions in support to free / open source software and related innovation. I was trained as a mathematician and theoretical computer scientist, and hold a Doctorat and the Habilitation à Diriger les Recherches from University Paris 7. From 1972 to 1981, I worked in software engineering research groups of software companies. I went as a research fellow to U.C. Berkeley in 1982. Since then, and before joining the European Commission in 1996, I headed research teams in the field of computer processing, indexing, retrieval and interaction for audiovisual media (video, music, still images). I am the author of more than 60 technical papers (stopped counting at some point), as well as of papers on the history, economy and sociology of information exchanges.

[1] Joseph E. Stiglitz, “La libéralisation a été programmée par les pays occidentaux pour les pays occidentaux”, Le Monde, 5 November 2001, http://www.lemonde.fr/rech_art/0,5987,239761,00.html (quote translated back

into English by the author).

2http://www-ce-faculty.stanford.edu/~knuth/faq.html

3See the article by John R. Koza on “Human-competitive machine intelligence by means of genetic programming” in the special insert on genetic programming in IEEE Intelligent Systems, May-June 2000, http://www.computer.org/intelligent/ex2000/pdf/x3074.pdf

4Jim Bessen and Eric Maskin, “Sequential Innovation, patents and imitation”, MIT and Harvard Working Paper Series, 00-01, January 2000 (http://www.researchoninnovation.org/patent.pdf) remains to my knowledge the main source of analysis at macro-level based on serious facts.

5This analysis is an excerpt from my paper “Positive Intellectual Rights and Information Exchanges”, to appear in Michael Century, ed. CODE, MIT Press, 2002.

6Cf Henri Atlan, *La fin du tout génétique*, INRA éditions, 1999, on genetic sequences and the fact that it is the set of the sequence and the complete cellular expression machinery that constitutes a biological process. For software, people are often confused by the reference to universal computers as being equivalent one to another, which would seem to support the fact that indeed a software "represents its execution". But as early as 1948, John von Neumann ("The general and logical theory of automata", Hixon.Symp. Lecture, Pasadena, in Collected Works 5:288-338, 1948) developed a luminous analysis of why this theoretical equivalence was practically not effective, and in any case it applies only to programs with predetermined input/output. In practice, as describes under question 1, a software technical effect can be understood only if its full execution environment is specified, including compilers, run-time, input-output devices, input-output contents.

7Academic researchers (and more generally innovators in software) usually treat feature patents by ignoring them. How would one do otherwise? A single software demonstrator can infringe on dozens of them, they are generally trivial and easy to invalidate by prior art. Even more, they are very rarely enforced aggressively except against industry competitors who have had the impertinence to start invading the dominant position of the patentee, which these feature patents were supposed to protect. There is little penalty for ignoring them, except potentially for those researchers who are also producers of free / open source software.

8Some software patent supporters have argued that there is no contradiction at all between patentability and open source software, and that patents could even be used to its benefit. This seems to ignore some major practical issues (entry cost, delays and incompatibility in philosophy and motivation). It should nonetheless be noted that some companies which are engaged in major open source efforts (IBM, Sun, Netscape, HP) are also big software patentees. The future interaction between these aspects is an open issue.

9Depending on which share of the category “Internet technology” is software.

10Source: “The Software sector: growth, structure and policy issues”, Working Party on the Information Economy, OECD, October 2001.

11Union Syndicale, Bulletin Agora, Juin 2000, pages 9-10 (translated from French by the author). On-line at: http://home.tvd.be/rc20042/public/

12http://www.european-patent-office.org/epo/facts_figures/facts2000/pdf/fact_figures_2000.pdf

13http://www.european-patent-office.org/epo/facts_figures/facts2000/pdf/fact_figures_2000.pdf

14http://www.european-patent-office.org/news/pressrel/2000_11_29_e.htm

15http://www.european-patent-office.org/news/pressrel/2001_10_05_e.htm

16http://europa.eu.int/comm/internal_market/en/indprop/softpaten.htm

17http://www.heise.de/newsticker/data/anw-24.10.01-004/

18USPTO also claims that it has already started going in the right direction by lowering from 56 to 36% the rate of accepted patents in the software-related categories.