SYMPOSIUM

20 October 2005

IPR on software: the road ahead

Centrum voor Wiskunde en Informatica

CWI in Bedrijf

Kruislaan 413, 1098 SJ, Amsterdam
www.cwi.nl/CiB

Symposium IPR on software: the road ahead

20 October 2005, 13.30 – 16.30h

After the recent rejection of the proposed directive on computer-implemented inventions by the European Parliament, the political turmoil and emotions surrounding the subject are cooling down. At the same time, the fundamental questions about intellectual property rights on software remain unanswered. This is therefore the right moment to resume the debate about the fundamentals of the patenting system and to explore the desirability and possibility to establish intellectual property rights (IPR) on software and to consider new systems to do so.
This symposium will contribute to identifying new ways to look at this old problem.

PROGRAMME

Chair: Maria Henneman

13.00	Arrival	Research demonstrations
13.30	Opening	Jan Karel Lenstra
13.35	Introduction	Paul Klint
13.45	Reinventing intellectual property protection for software	Robert Plotkin
14.25	Computer-implemented inventions: the EPO legal framework and practice	Yannis Skulikaris
15.00	Tea break	Research demonstrations
15.20	A patented method to fix the patent system	Roland Orre
15.40	Discussion	Jan Bergstra, Lucie Guibault, Roland Orre, Robert Plotkin, Yannis Skulikaris
16.30	Closure Drinks	Jan Karel Lenstra Research demonstrations

IPR on software: the road ahead – 20 October 2005

ABSTRACTS

Paul Klint:

IPR on software: the road ahead

After the recent rejection of the proposed directive on computer-implemented inventions by the European parliament, the political turmoil and emotions surrounding the subject of software patents are cooling down. At the same time, the fundamental questions about intellectual property rights (IPR) on software remain unanswered. This is therefore the right moment to resume the debate about the fundamentals of the patenting system and to explore the desirability and possibility to establish IPR on software and to consider new systems todo so. In this half-day symposium, the following questions will be addressed by the invited speakers and panellists:

Is there anything about software that makes it 'special'and justifies a special treatment under regimes of IPR protection?
Can we learn from the role of IPR in other disciplines like, for instance, the life sciences?
Patents are about the protection of inventions; what is the character of inventions in the software field?
Should all software be treated the same or should we distinguish different categories (e.g., embedded software versus non-embedded software) for IPR protection?
S-called 'trivial patents are one of the major stumbling blocks of the current patenting system: how can they be avoided?
How can we build up knowledge about 'prior art' to protect against such patents?
Are there alternatives for the current IR protection mechanisms(copyright, patent) thatcould work better for software?
How should we look at the role of IPR in relation to new players such as China and India?

This symposium aims at contributing to identifying new ways to look at this old problem.

Robert Plotkin:

Reinventing intellectual property protection for software

The debate over whether software is appropriate subject matter for patent protection continues to rage, without any clear resolution in sight. The European Patent Convention attempts to draw this distinction by excluding software programs from patent protection unless they have a 'technical effect'. U.S. law requires that a software program have a 'practical utility' to constitute patentable subject matter. Both such requirements fail to provide clear guidance in hard cases because they merely beg the question of what constitutes a 'technical' effect or a 'practical' utility. I recommend that software patent claims instead be evaluated in terms of three kinds of utility for effects: 1) *physical* utility, which refers to the direct physical effects of an invention, such as the movement of gears and levers in a mechanical cash register; 2) *logical* utility, which refers to the information processing tasks performed by an invention, such as the mathematical operations performed by a cash register; and 3) *application* utility, which refers to the usefulness of the invention to the end user, such as the increased accuracy and reduced calculation time made possible by a cash register. In particular, the physical utility requirement would require the applicant to demonstrate that the claimed software is susceptible of embodiment in a tangible form,

the logical utility requirement would require the claimed software to be defined clearly in terms of the information processing steps it performs, and the application utility requirement would require the applicant to provide an industrial application of the claimed software. Furthermore, patent examiners and judges should be required to evaluate software patent claims expressly in terms of each kind of utility. This would clarify the basis for the decision in each case and avoid the now-common conflation of one kind of utility with another, thereby focusing any dispute over subject matter patentability on the particular kind of utility whose satisfaction is in question.

Yannis Skulikaris:

Computer-Implemented Inventions: the EPO legal framework and practice

There is a lot of controversy regarding Computer-Implemented Inventions (CII), and there are good reasons for that: the special nature of software, the exclusion of patentability of computer programs as such in the European Patent Convention, the different Intellectual Property systems on the two sides of the Atlantic, the effort to harmonize patent law and practice within Europe etc.

From an economy perspective, the Open Source model claims to be inherently incompatible with CII patents. And SME's express fears that they might be crushed under the IP rights of the IT giants. In an environment where patent rights have to be balanced with the requirement of free competition, the European Patent Office operates in a very special way: being exclusively an executive organ, the EPO follows the European Patent Convention and the case law, but still detects the signals coming from the users and those indirectly affected by the patent system.

Roland Orre:

A patented method to fix the patent system

The patent system has flaws. Flaws can be abused or they can be used against the system to improve the system.

The crazy idea to patent a method which could fix the flaws in the patent system occurred as a thought experiment inspired by a debate about software patents early 2000. The idea was to illustrate how ridiculous the concept of patents on business methods and software was.

Later I realised, that it is actually possible to implement this idea, and also, that such a patent, when thoroughly worked out as a well founded business idea, could have very beneficial effects on the society and the economy.

The patent is an artificial intelligence method applied to business, with potential to accelerate the technological evolution, counteract the 'Adam Smith' effect, to prepare the society and human mind for nano technology, and push the patent system away from trivial patents and software patents towards real innovations.

However, the somewhat controversial effect is that it implies that patents on software and business method patents are not completely evil, as the beneficial effects of this business method would certainly be hard to achieve, without these flaws of the system .....

ABOUT THE SPEAKERS

Jan Bergstra is professor of programming and software engineering at the Universiteit van Amsterdam and professor of applied logic at Utrecht University, Department of Philosophy. Earlier he worked at Leiden University, CWI, and Philips Research. Since 2001 he is director of the Educational Institute of Information Sciences at Universiteit van Amsterdam.
Jan Bergstra's research has always been oriented towards issues of computation and computability. After having been chairman of the organizing committee of the first Holland Open Software Conference (HOSC 2005, Amsterdam) Jan Bergstra got unexpectedly involved in research on software patents together with Paul Klint. This work is now being continued in the form of a consultancy project for the European Commission.

Lucie Guibault is assistant professor of copyright and intellectual property law at the Institute for Information Law (IVIR) of the Universiteit van Amsterdam. Born and raised in Canada, she studied law at the Université de Montréal (LL.B. 1988 and LL.M. 1995) and recently received her doctorate from the Universiteit van Amsterdam where she defended her thesis on Copyright Limitations and Contracts: An Analysis of the Contractual Overridability of Limitations on Copyright. She joined the Institute for Information Law in 1997.
Dr. Guibault is specialized in international and comparative copyright and intellectual property law; she is in charge of the coordination of the International Copyright Law Summer Course, given by IVIR staff at the Amsterdam Law School.

Paul Klint is head of the software engineering department at Centrum voor Wiskunde en Informatica (CWI), - the Dutch national research centre for computer science and mathematics, professor of computer science at the University of Amsterdam, and visiting professor at the University of London (Royal Halloway). He is also president of the European Association for Programming Languages and Systems (EAPLS) and co-founder of the Software Improvement Group (SIG), a CWI spinoff company. He holds an MSc in Mathematics from the University of Amsterdam (1973) and a PhD in Computer Science from the Technische Universiteit Eindhoven (1982). He is currently directing the one-year Master's Programme in Software Engineering at the Universiteit van Amsterdam. His research interests include generic language technology, domain-specific languages, component technology, software renovation, technology transfer and intellectual property rights. He has consulted for companies and governments worldwide.

Jan Karel Lenstra is General Director of Centrum voor Wiskunde en Informatica (CWI), the Dutch national institute for research in computer science and mathematics. He has been on the research staff of CWI and on the faculty of the Technische Universiteit Eindhoven – where he served as Dean of Mathematics & Computer Science – and of the Georgia Institute of Technology, Atlanta. His research interests are in the combinatorial optimization, in particular sequencing and scheduling, routing, complexity, approximation and local search.

Roland Orre was trained in Engineering Physics and received a doctorate in computer science at the Royal Institute of Technology (Sweden); is involved in collaborative research and development with WHO, developing early warning signaling methods, as well as unsupervised pattern recognition and duplicate report detection. Founded company NeuroLogic Sweden AB and is CEO. Patents: US+PCT patent application – applied, (final appl. 10th March 2005, prio 10th March 2004) for a patent on a data mining method which turns a search engine into a (semi)automatic manufacturing system.
EPO patent – *A method of extracting a logical description of a sampled analog signal.*
Ref: http//12.espacenet.com/espacenet/viewer?PN=EPO520400

Robert Plotkin, Esq. is an attorney specializing in intellectual property protection for computer hardware and software. He has an extensive background in computer science shared by few other attorneys. He began programming as a hobby at the age of ten and has devoted himself to the study of computer science and the development of computer technologies ever since. He has a Bachelor's degree in Computer Science and the Engineering from the Massachusetts Institute of Technology and continually updates his knowledge of the latest developments in the field of computer science both through his client work and through his memberships in professional associations, such as the Association for Computing Machinery (ACM) and the Institute for Electrical and Electronics Engineers (IEEE).
Attorney Plotkin is also a lecturer at the Boston University School of Law, where he teaches an advanced course titled ' Software and the law'. He has written and spoken extensively on topics including software patent protection, First Amendment protection for computer source code, trademark and domain name disputes, patent licensing, electronic court filing, and electronic privacy in the work place.

Yannis Skulikaris has a background in Physics, Computer Science and Law. He has worked for eight years in the European IT sector, first as a programmer in the Greek public administration, then as an office automation consultant with the British Petroleum in Hamburg. He has also worked as a freelance IT consultant. He has been with the European Patent Office since 1989, first as a patent examiner and then as a manager. He is currently heading a directorate of 30 patent examiners working in the field of computer-implemented inventions (CII) at the EPO branch in The Hague. He has been deeply involved as a technical expert in the recent process concerning the CII directive.

Maria Henneman, well known as a broadcasting journalist, is a media/communications consultant.

Software Patentability and Practical Utility: What's the Use?

© 2004 by Robert Plotkin, Esq.
rplotkin@rplotkin.com
Robert Plotkin, P.C.
Concord, MA USA

This is a preprint of an article whose final and definitive form has been published in the International Review of Law Computers and Technology. 2005 Copyright Taylor & Francis; International Review of Law Computers and Technology, is available online at http://taylorandfrancis.metapress.com/link.asp?id651301m2x162p1.

Abstract

The debate over whether software is appropriate subject matter for patent protection continues to rage, without any clear resolution in sight. The European Patent Convention attempts to draw this distinction by excluding software programs from patent protection unless they have a "technical effect." U.S. law requires that a software program have a "practical utility" to constitute patentable subject matter. Both such requirements fail to provide clear guidance in hard cases because they merely beg the question of what constitutes a "technical" effect or a "practical" utility. I recommend that software patent claims instead be evaluated in terms of three kinds of utility or effects: (1) *physical* utility, which refers to the direct physical effects of an invention, such as the movement of gears and levers in a mechanical cash register; (2) *logical* utility, which refers to the information processing tasks performed by an invention, such as the mathematical operations performed by a cash register; and (3) *application* utility, which refers to the usefulness of the invention to the end user, such as the increased accuracy and reduced calculation time made possible by a cash register. In particular, the physical utility requirement would require the applicant to demonstrate that the claimed software is susceptible of embodiment in a tangible form, the logical utility requirement would require the claimed software to be defined clearly in terms of the information processing steps it performs, and the application utility requirement would require the applicant to provide an industrial application of the claimed software. Furthermore, patent examiners and judges should be required to evaluate software patent claims expressly in terms of each kind of utility. This would clarify the basis for the decision in each case and avoid the now-common conflation of one kind of utility with another, thereby focusing any dispute over subject matter patentability on the particular kind of utility whose satisfaction is in question.

I. Framing the Problem of Software Patentability

The debate over software patentability continues to rage, without any clear resolution in sight, after at least four decades of controversy.[1] The debate over software "patentability" has tended to focus, quite reasonably, on the threshold "subject matter patentability" requirement,[2] which specifies the kinds of subject matter that are susceptible to patent protection. The patentable subject matter requirement is an attempt to provide a mechanism for filtering out claimed subject matter which does not fall within the "useful" arts (in the U.S.) or the "industrial" arts (in Europe). Failure to satisfy the patentable subject

[1] The debate dates back at least to 1965, when U.S. President Johnson commissioned a comprehensive study of the United States patent system. The Commission explored a wide range of pressing issues facing the patent system and recommended, among other things, that "no patents on . . . computer programs" be issued. 1996 Report of the President's Comm'n on the Patent Sys. 1.

[2] In the U.S., patentable subject matter is defined in 35 U.S.C. § 101. In the European patent system, patentable subject matter is defined in Article 52 of the European Patent Convention.

matter requirement is fatal to the ultimate determination of patentability more generally. A poem, for example, is not patentable subject matter and therefore not patentable, no matter how novel it may be and no matter how significant an advance it represents over the current state of the literary art.

The U.S. patent statute adopts a categorical approach to patentable subject matter, according to which five categories of subject matter –processes, machines, articles of manufacture, and compositions of matter –are susceptible to patent protection.[3] The European Patent Convention takes an approach which is at once more expansive and more limiting, by establishing "industrial application" rather than a list of specified categories as the fundamental test for patentable subject matter, while also providing a list of expressly excluded categories.[4] Although this express exclusion includes computer programs, computer programs are only excluded "as such."[5]

In most cases, the question, "is software patentable?," is best interpreted as the question, "is software patentable subject matter?" Although other requirements, such as the novelty[6] and inventive step/nonobviousness[7] requirements, have also engendered controversy in their application to software, the crux of the debate has been and continues to be whether computer programs, as a class, should even be susceptible to patent protection in the first instance.[8]

II. The Failure of Existing Standards

The current state of the law of software subject matter patentability, both in the U.S. and Europe, leaves much to be desired, particularly considering the volume of ink that has been spilled in consideration of the topic. In particular, although some consensus has been reached that software can qualify as patentable subject matter under certain conditions, we lack clear and objective definitions of such conditions, and therefore lack the means to determine whether any particular software patent claim defines patentable subject matter. For example, although a computer program that merely performs a mathematical calculation lacking any practical application does not qualify as patentable subject matter,[9] while software that controls an automobile-manufacturing robot likely qualifies as patentable subject matter, even such conclusions are less certain than is desirable and leave a substantial grey area in-between.

The European Patent Office (EPO) Board of Appeal and the EPO Guidelines for Examination distinguish between patentable and non-patentable subject matter in the context of software based on whether the claimed subject matter has a "technical character" or "technical effect."[10] The Court of Appeals for the Federal Circuit in the U.S. has effectively discarded the categorical approach of the U.S. patent statute in favor of a "practical utility" requirement,[11] according to which claimed subject matter must achieve a "useful, concrete, and tangible result" to qualify as patentable subject matter.[12]

These reinterpretations of the patentable subject matter requirement are steps in the right direction. For example, both the "technical effect" and the "practical utility" requirement provide a basis for excluding indisputably non-patentable subject matter, such as abstract ideas, laws of nature, and natural phenomena.[13] Both the European and U.S. approaches, however, fail to provide clear guidance in hard

---

[3] 35 U.S.C. § 101.
[4] Art. 52(2) Eur. Pat. Convention (Oct. 5, 1973).
[5] *Id.* 52(3).
[6] *Id.* 54(1); 35 U.S.C. § 102.
[7] *Id.* 56; 35 U.S.C. § 103.
[8] For an overview of the history of the debate over software subject matter patentability, *see, e.g.*, GREGORY A. STOBBS, SOFTWARE PATENTS 1-46 (2d ed. 2000).
[9] *See, e.g.*, Diamond v. Diehr, 450 U.S. 175 (1981).
[10] Guidelines for Examination in the EPO § C-IV.2 (2001).
[11] State Street Bank & Trust Co. v. Signature Financial Group, Inc., 149 F.3d 1368, 1375 (Fed. Cir. 1998).
[12] Id.
[13] Diamond v. Diehr, 450 U.S. at 185.

cases because they merely beg the question of what constitutes a "technical" effect in contrast to a "non-technical" effect, or a "practical" utility in contrast to a "non-practical" utility.

Every computer program executing on a computer produces a "technical" effect in the sense that execution of the program causes the computer to store, modify, and transmit electrical signals internally. If the mere manipulation of electrical signals by a computer as an inherent part of executing a computer program were sufficient to confer subject matter patentability on the program, then every computer program would satisfy the "technical effect" standard. Assuming that at least some computer programs, such as those which merely calculate the results of a mathematical formula having no practical application, should not qualify as patentable subject matter, the "technical effect" standard would fail as a useful test for subject matter patentability if "technical effect" merely required the performance of physical activity by a machine.[14]

One might be tempted to fix this problem of overinclusiveness by requiring a "technical effect" beyond the internal manipulation of electrical signals that is inherent to the execution of any computer program.[15] Such a requirement of "technical effect plus," while solving the overinclusiveness problem, would swing too far in the opposite direction by excluding from subject matter patentability a variety of worthy software.

Consider, for example, a hypothetical case involving two black boxes which are identical in external appearance and behavior. Each box has an input slot into which an original x-ray print may be fed. After a short delay, each box produces a highly clarified x-ray print in which any tumors are highlighted. The clarified x-rays produced by both boxes are indistinguishable from each other. Assume that the quality of x-ray clarification produced by both boxes is better than that which may be obtained using any preexisting x-ray processing device.

Upon opening both boxes and peering inside, you find in the first box a complex jumble of circuitry and are informed that such circuitry was custom-designed by an electrical engineer. In the other box you find a small laptop computer running x-ray image processing software written by a computer programmer. The circuitry in the first box and the software in the second box implement precisely the same x-ray clarification algorithm.

Is there any basis for deeming the circuit-implemented x-ray clarification device to constitute patentable subject matter, but not the software-implemented clarification device? I assert not; the subject matter patentability of the circuit-based and software-based implementations must stand or fall together. One may consider this thought experiment to be akin to a "Turing test" for patentable subject matter, in the sense that the subject matter patentability determination must be made based on the externally-observable behavior of the device in question, rather than on any knowledge of whether the device is implemented in hardware or in software.

There should be no question that the circuit-implemented x-ray clarification device in the hypothetical example above would qualify as patentable subject matter under any version of the "technical effect" test. Therefore, the software-implemented device must also qualify as patentable subject matter. Under a strict interpretation of the "technical effect plus" requirement, however, the software-implemented device would not qualify as statutory subject matter because its execution on the computer merely involves the manipulation of electrical signals inherent to the execution of any computer program. Any attempt to loosen the "technical effect plus" requirement, such as by allowing the production of the clarified x-ray print to qualify as the "plus," would create an exception that would swallow the rule. Any computer program, including those which do not have any practical/industrial application and which therefore should not qualify as patentable subject matter, may be modified to produce output external to the computer. The "technical effect plus" standard, therefore, whether construed strictly or

---

[14] As described in more detail below in Section III.A.1, this naïve "physicality" test for statutory subject matter has been adopted in a variety of European and U.S. cases.

[15] For an example of such an approach, *see, e.g.*, Koch & Sterzel/X-ray Apparatus (T26/86, Eur. Pat. Office J. 1-2 1988).

liberally, fails to provide a useful basis for distinguishing between patentable subject matter and non-patentable subject matter.

The "practical utility" requirement suffers from similar shortcomings. If "practical" equates to "physical," then all software executing on a computer has "practical" utility in the sense that all software creates physical effects within the computer in the form of electrical signal manipulation. For example, under such a test, digital music would (incorrectly) qualify as patentable subject matter. Therefore, "practical utility" must require something beyond mere internal "physicality." Requiring some physical effect external to the computer, however, would fail to provide a useful standard for the reasons described above with respect to the "technical effect plus" standard.

The Court of Appeals for the Federal Circuit (CAFC), perhaps recognizing the limitations of a naïve physicality test for subject matter patentability, looked beyond mere physicality to evaluate the practical utility of the claimed software in *State Street Bank & Trust Co. v. Signature Financial Group, Inc.*[16] In particular, the CAFC held that:

> the transformation of data, representing discrete dollar amounts, by a machine through a series of mathematical calculations into a final share price, constitutes a practical application of a mathematical algorithm, formula, or calculation, because it produces "a useful, concrete and tangible result"—a final share price momentarily fixed for recording and reporting purposes and even accepted and relied upon by regulatory authorities and in subsequent trades.[17]

This passage, although much-criticized for its use in eliminating the business method exception to subject matter patentability, is commendable for its ability to look beyond the physical implementation details of the financial software in question to the "real-world" utility of the software, whatever one may think of the conclusion drawn. An analysis which takes into account the real-world benefits that a claimed program provides to its users represents an advance over an approach which focuses solely on whether the claimed program generates electrical signals within a computer.

Unfortunately, the CAFC was unable to sustain its progress, however incremental, for more than one case. In *AT&T Corp. v. Excel Communications, Inc.*,[18] the CAFC eliminated physicality as a requirement for subject matter patentability.[19] Elimination of any physicality requirement is inconsistent with both precedent[20] and the fundamental patent policy of only protecting innovations in the useful/industrial arts.[21] For example, the elimination of the physicality requirement leaves no clear basis for excluding from subject-matter patentability either abstract ideas[22] or a variety of works falling solely within the liberal arts (such as purely human-performed innovations in law, politics, and business).

In summary, recent jurisprudence in both Europe and the U.S. has reframed the question of software subject matter patentability in terms of "technical effect" and "practical utility." These standards, however, have significant practical and theoretical limitations. As a result, a coherent standard still is lacking for determining whether a claimed computer program constitutes patentable subject matter.

III. Three Utilities for the Price of One

Analysis of opinions in software patent cases reveals that patent examiners and judges actually evaluate software patent claims in terms of three kinds of interrelated utility: physical utility, logical utility, and application utility. "Physical utility" refers to the direct physical effects of an invention, such as the

---

[16] 149 F.3d 1368 (Fed. Cir. 1998).
[17] *Id.* at 1373.
[18] 172 F.3d 1352 (Fed. Cir. 1999).
[19] *Id.* at 1358-60.
[20] *See, e.g.*, Cochrane v. Deener, 94 U.S. 780, 788 (1876) (holding that "[a] process is . . . an act, or a series of acts, performed upon the subject-matter to be transformed and reduced to a different state or thing").
[21] U.S. CONST. art. I, § 8, cl. 8; Art. 52(1) Eur. Pat. Convention (Oct. 5, 1973).
[22] *See supra* n.13.

movement of gears and levers in a mechanical cash register. "Logical utility" refers to the information processing tasks performed by an invention, such as the mathematical operations performed by a cash register. "Application utility" refers to the "real-world" function performed by the invention from the perspective of the end user, such as the calculation by a mechanical cash register of the total cost of a bag of groceries.

Although patent examiners and judges already consider these three kinds of utility, they tend to do so implicitly rather than explicitly, without explaining the basis (if any) for evaluating particular claims in terms of one kind of utility rather than another, and with a lack of consistency from case to case. In the following discussion I provide examples[23] of such confusion and explain how it leads to inconsistent and unprincipled results.

A. Case Studies

1. Physical Utility

In some cases, the subject matter patentability determination has hinged on the "physicality" of the claimed subject matter, e.g., whether a claimed process or product is implemented in a physical form and/or acts on physical material. In the U.S., the seminal case of *Cochrane v. Deener* established a physicality requirement for processes in the form of the proposition that a process only qualifies as patentable subject matter if it transforms something "into a different state or thing."[24]

In *In re* Sherwood,[25] the Court of Claims and Patent Appeals (CCPA) held claims directed to techniques for seismic prospecting to constitute patentable subject mater on the grounds that "seismic traces [recited in the claims] are electrical signals from geophones, i.e., *physical* apparitions, or particular patterns of magnetization on magnetic tape, i.e., the pattern of the magnetization being a *physical* manifestation, or a *physical* line on a paper chart." In *In re* Walter,[26] the CCPA distinguished between inventions in which "the end product . . . is a pure number," and which therefore are non-patentable subject matter, and inventions which "produce[] a physical thing," which may be patentable subject matter even if the physical thing is represented in numerical form. In Diamond v. Diehr,[27] the U.S. Supreme Court held a process to constitute patentable subject matter because the claims "involve[d] the transformation of . . . raw, uncured synthetic rubber, into a different state or thing." In *In re* Pardo,[28] the CCPA held that a process claim directed to controlling the internal operations of a programmed computer constituted statutory subject matter because the claim was directed to "executing programs in a computer," which the court viewed as indistinct from a strictly mechanical adding machine. In *In re* Grams,[29] the CAFC held that an algorithm that failed to perform "physical steps" did not qualify as statutory subject matter.

In *State Street Bank & Trust Co. v. Signature Financial Group, Inc.*,[30] the CAFC held that "the transformation of data, representing discrete dollar amounts, by a machine through a series of mathematical calculations into a final share price," qualified as statutory subject matter because it produced "a useful, concrete and tangible result," even though the transformed data did not represent physical activity or objects. Although the court in *State Street* did not expressly refer to the *physical* transformations performed by the claimed machine, the phrase "transformation of data" must refer to a transformation of physical electrical signals because the machine that performed the transformation in

---

[23] The following discussion does not provide an exhaustive set of case studies, but rather provides illustrative examples.
[24] 94 U.S. 780, 788 (1876).
[25] 613 F.2d. 809, 819 (C.C.P.A. 1980).
[26] 618 F.2d 758, 767-68 (C.C.P.A. 1980).
[27] 450 U.S. 175, 185 (1981).
[28] 684 F.2d 912, 916 (C.C.P.A. 1982).
[29] 888 F.2d 835, 840 (Fed. Cir 1989).
[30] 149 F.3d 1368, 1373 (Fed. Cir. 1998).

State Street was a programmed digital electronic computer.[31] The court's holding may therefore be interpreted, in its best light, to stand for the proposition that a machine which performs a *physical* transformation (e.g., of electrical signals from one form into another) qualifies as statutory subject matter if the result of the transformation is useful, concrete, and tangible (i.e., physical).

In Vicom/Computer-Related Invention,[32] the Board of Appeal of the European Patent Office held claims directed to a process and product for digital image processing to satisfy the patentable subject matter requirement, reasoning that "if a mathematical method is used in a technical process, that process is carried out on a physical entity (which may be a material object but equally an image stored as an electrical signal) by some technical means implementing the method and provides as its result a certain change in that entity."[33] In IBM/Data Processor Network,[34] claims related to the coordination and control of the internal communications between programs and data files stored in different computers connected as nodes in a telecommunications network were held to satisfy the patentable subject matter requirement because the invention was concerned with the internal workings of computer processors.

In summary, in a variety of cases, physical utility has been a basis, and in some cases the sole basis, for granting or denying subject matter patentability. In many such cases, the logical or application utility of the claimed subject matter has been at most a secondary consideration.

2. Logical Utility

Other cases have focused primarily on the logical utility of the claimed software, i.e., on the algorithm or other information processing steps recited in the claim, independently of the physical instantiation of the program (physical utility) or its higher-level real-world use (application utility).

As mentioned above, in *AT&T Corp. v. Excel Communications, Inc.*,[35] the CAFC eliminated physicality entirely as a requirement for subject matter patentability,[36] stating that "a mathematical algorithm may be an integral part of patentable subject matter such as a machine or process if the claimed invention as a whole is applied in a 'useful' manner."[37] By focusing not on the physical implementation of the claimed method but rather on the manner in which it processed data, the CAFC in effect evaluated the claimed method at least in part based on its logical utility.

In IBM/Document Abstracting and Retrieving,[38] the EPO Board of Appeal rejected a claim directed to abstracting a document, storing the abstract, and retrieving it in response to a query, on the ground that the electrical signals generated when implementing the claim do not represent a physical thing, but rather represent part of "the information content of a document, which could be of any nature."[39] By evaluating the claim based on the *information* represented by the process, the Board of Appeal in effect evaluated the claim based on its logical utility.

The opinions in some cases have been at best unclear about whether satisfaction of the patentable subject matter requirement hinges on physical utility, logical utility, or some combination of both. This confusion most commonly manifests itself in lack of clarity about whether the patentable subject matter requirement requires a claimed product or process to actually transform physical material into a different state or thing, or whether it is sufficient that the product or process in question manipulate data *representing* physical material. The former would correspond to the "physical utility" requirement

---

[31] *Id.*
[32] 2 Eur. Pat. Office Rep. 74 (1987).
[33] *Id.* at 79.
[34] T6/83, Eur. Pat. Office J. 1-2 (1990).
[35] 172 F.3d 1352 (Fed. Cir. 1999).
[36] *Id.* at 1358-60.
[37] *Id.* at 1357.
[38] T115/85, Eur. Pat. Office J. 1-2 (1990).
[39] *Id.* at para. 13.

described herein, while the latter would correspond most closely to the "logical utility" requirement, due to its focus on the information processing steps performed by the claimed product or process.

For example, in *In re* Taner,[40] the CCPA held that claimed processes which both *performed* and *simulated* physical activity satisfied the patentable subject matter requirement, without distinguishing between these two senses of physicality or explaining the relevance of either to the subject matter patentability determination. In *In re* Abele,[41] the CCPA based its patentable subject matter determinations on whether the data used by the claimed processes *represented* physical entities. In *In re* Schrader,[42] the CAFC held the disputed process claims not to be directed to statutory subject matter because they "do not reflect any transformation or conversion of subject matter *representative of or constituting* physical activity or objects." In this passage, it is not clear whether the "physicality" requirement requires a claimed process to be implemented in ("constitute") physical activity or objects, or merely to perform operations that are "representative of" physical activity or objects.[43]

3. Application Utility

Finally, in certain cases, the primary focus has been on the application, or "real-world," utility of the claimed software, independently of whether the software effects physical transformations or performs particular information-processing steps. For example, the CAFC relied on application utility in *State Street* when it found the disputed claim to satisfy the patentable subject matter requirement because the claimed system produced "a final share price momentarily fixed for recording and reporting purposes *and even accepted and relied upon by regulatory authorities and in subsequent trades*" (emphasis added).[44] Such reasoning focuses on the benefits provided by the invention both to its direct users and to indirect beneficiaries (e.g., regulatory authorities), regardless of the particular physical form in which the invention is instantiated or the particular information processing steps performed by the invention.

In IBM/Semantically Related Expressions,[45] a claim to a text processing system for automatically generating semantically-related expressions was rejected on the ground that the system belonged to the field of linguistics.[46] According to such an approach, whether a particular invention qualifies as patentable subject matter depends on whether the *field* of the invention falls within the industrial arts or the liberal arts, again regardless of the particular physical form in which the invention is instantiated or the particular information processing steps performed by the invention.

B. Recommendation: Require All Three Kinds of Utility

The case studies provided above demonstrate that although patent claims routinely are evaluated in terms of physical, logical, and application utility, these three different kinds of utility have not expressly been recognized as such. Rather, patent examiners and judges appear to conflate the different kinds of utility with each other and to select different kinds of utility arbitrarily as the basis for the subject-matter patentability determination from case to case. As a result, no coherent rules have emerged for evaluating software patent claims in terms of physical, logical, and application utility.

---

[40] 681 F.2d 787, 790 (C.C.P.A. 1982).
[41] 684 F.2d 902 (C.C.P.A. 1982).
[42] 22 F.3d 290, 294 (Fed. Cir. 1994).
[43] Arrhythmia Research Technology v. Corazonix Corp., 958 F.2d 1053 (Fed. Cir. 1992) is another example of a case which repeatedly conflates the two senses of "physicality" noted herein. For further discussion of confusing surrounding these two senses of "physicality," *see* Robert Plotkin, *Computer Programming and the Automation of Invention: A Case for Software Patent Reform*, 2003 UCLA J.L. & Tech. 7 (2003), § III.D.5.
[44] *Id.* at 1373.
[45] T52/85 Eur. Pat. Office J. R-8, 454 (1989).
[46] *Id.* at 458.

I recommend that software patent claims be required to satisfy all three utility requirements. The burden should be on the software patent applicant to demonstrate that the claimed invention has all three kinds of utility.

In particular, the physical utility requirement should require the applicant to demonstrate that the claimed software is susceptible of embodiment in a tangible form. In effect, this requirement would serve the same purpose as the enablement requirement:[47] to ensure that the applicant has taught the public to make and use the claimed invention. The physical utility requirement, therefore, would simply require that the patent disclosure enable one of ordinary skill in the art to implement the claimed program on a computer.[48] The physical utility requirement would, therefore, impose a physicality requirement that would provide an initial layer of protection against granting patent protection to claimed subject matter falling outside the technological and/or industrial arts.

The logical utility requirement would require software patent claims to clearly and particularly define the information processing steps performed by the claimed software. Existing rules of claim construction could be adapted to ensure that claims drafted too broadly would either not qualify as statutory subject matter or be interpreted narrowly.[49] Software patent claims which merely claimed the result achieved by a software program, for example, would not satisfy the logical utility requirement. Such a requirement would both help to ensure that the patentee has put the public on notice of what is claimed and to limit the scope of the claims to the scope of enablement.

Finally, the application utility requirement would require the patent applicant to disclose a specific, substantial, and credible technological (in the U.S.) or industrial (in Europe) application of the claimed software.[50] This requirement, therefore, would focus on the end use, or "real-world" utility, of the claimed software. A claim to a computer program for calculating the results of a novel and nonobvious mathematical formula, for example, would satisfy the physical utility requirement (assuming that the patent specification adequately described how to implement the program) and the logical utility requirement (assuming that the patent specification and claims defined with sufficient clarity and specificity the steps required to evaluate the formula), but not the application utility requirement absent recitation of a specific, substantial, and credible application of the formula. The application utility requirement would thereby provide an additional layer of protection against granting patent protection to developments falling outside the technological/industrial arts.

Patent examiners and judges should be required to evaluate software patent claims expressly in terms of each kind of utility when crafting their opinions. This would clarify the basis for the decision in each case and avoid the above-described and now-common conflation of one kind of utility with another, thereby focusing any dispute over subject matter patentability on the particular kind of utility whose satisfaction is in question. Clarifying the basis for subject-matter patentability determinations would both reduce the degree to which the applicant and examiner argue past each other and clarify issues for appeal.

Another benefit of the approach proposed herein is that the three recommended utilities are not selected arbitrarily, but rather represent points on a continuum ranging from low-level physical activity at one end to high-level social function at the other end.[51] Although utilities at additional and/or alternative

---

[47] 35 U.S.C. § 112 ¶ 1; Art. 83 Eur. Pat. Convention (Oct. 5, 1973).

[48] Note, however, that a disclosure of a computer program may be enabling without describing the program in physical terms. *See* Plotkin, § III.D.

[49] For a more detailed proposal for such modified rules of claim construction, *see* Plotkin § V.C.

[50] The requirements of specific, substantial, and credible utility are drawn from the United States Patent and Trademark Office Utility Examination Guidelines, 66 Fed. Reg. 1092 (2001), and represent a reasonable attempt to provide standards for evaluating whether an invention has a "real-world" utility.

[51] Physical utility arguably takes the "external perspective," while logical utility and application utility arguably take the "internal perspective" described in Orin S. Kerr, "The Problem of Perspective in Internet Law," 91 Georgetown Law Journal 357 (2003).

points on the spectrum could be selected,[52] the three utilities described herein closely reflect the utilities analyzed in actual cases[53] and serve as good exemplars for explaining operation of the theory proposed herein.

IV. Conclusions

The pragmatic solution proffered herein, while clarifying the analysis of utility in particular cases and providing some additional objective bases for substantive evaluation of the three kinds of utility, still begs the ultimate question: which kinds of utility satisfy the newly-created "application utility" requirement? For example, assuming that a patent claim to a software-implemented business method satisfies the physical and logical utility requirements, does the "business method" aspect of the software satisfy the "application utility" requirement? The approach outlined herein does not provide an answer to this ultimate question, although it does help to make clear that this is in fact the question whose answer is in dispute, rather than some other question about whether, for example, the claimed software is capable of being instantiated in a physical form.

One reason that such question-begging is so difficult, if not impossible, to avoid in this field of inquiry is that the meaning of "technology"[54] is ambiguous, dynamic, and value-laden.[55] Frederick Ferré, for example, has defined "technology" as the "practical implementations of intelligence,"[56] while Robert McGinn has defined "technological activity" as a "purposive, methodological enterprise that fabricates or is constitutive of material outcomes."[57] Such definitions not only fail to provide guidance in answering the question of software subject-matter patentability, they make clear exactly why the question is so difficult to answer.

Computers have expanded, both vastly and rapidly, the range of "intelligence" and "purposive, methodological enterprise" that may be implemented in automated procedures in the form of software. In general, patent law has tended to absorb new technological fields, such as those created by the harnessing of electricity and the invention of the internal combustion engine, relatively easily due to the fact that such new fields provided improved means for performing functions already within the province of technology. For example, although the internal combustion engine enabled substantial automation of transportation, patent law was well-versed with existing, albeit more primitive, transportation devices.

Although many kinds of software similarly automate or otherwise improve upon the functions performed by previous machines (as exemplified by a software word processor in comparison to an electric typewriter), computers also make it possible to implement automated procedures in the form of software that performs functions never previously performed by machines. Examples of such software include software for performing fully-automated business methods, natural language processing (e.g., speech recognition and language translation), and digital image processing. Similarly, expert systems, graphical user interfaces, and genetic algorithms enable computers to make decisions, interact with humans, and solve problems in ways never before achievable by machines alone.

---

[52] For example, "application utility" could further be subdivided into different levels of social utility, such as the utility of a process to its direct user and the higher-level utility of the process to an enterprise employing many such users performing many instances of the process. Returning the mechanical cash register example, the cash register's "user utility" might be the calculation of the total price of a bag of groceries, while the "enterprise utility" might be increased throughput of customers through the queue.

[53] *See supra* Section III.A.

[54] The following discussion is also applicable, though perhaps with a somewhat different flavor, to the meaning of "industrial."

[55] For an excellent discussion of this topic, *see* John R. Thomas, *The Post-Industrial Patent System*, 10 Fordham Intell. Prop. Media & Ent. L.J. 346 (1999).

[56] *Id.* at n.193.

[57] *Id.* at n.207.

The existence of software that performs functions previously requiring human judgment and even expertise, such as expert systems, illustrates that computers are expanding the range of activities that can be implemented in machines. In this sense, computers are merely doing what technology has always done. Books replaced bards and electromechanical looms replaced seamstresses, just as expert systems have begun to replace their human counterparts today. Although technological advances may not fully replace human skill in particular instances, there is no question that the goal of technology is to put the genie in the bottle. The problem raised by software is that the rate and scope of genie bottling has increased more rapidly than the law has been able to adapt.

In particular, software now makes it possible for functions previously classified within the *liberal arts* to be performed fully automatically by machines. Language translation software, image processing software, and software for performing fully-automated business methods are just a few examples. Law firms are already charging clients for use of software that provides legal services.[58] To the extent that the difference between patentable and non-patentable subject matter rests on a distinction between the technological arts and the liberal arts, the distinction breaks down in the face of the ability of a *technological device* – a general-purpose computer programmed with particular software – to perform functions traditionally falling within the *liberal arts*.[59] Although one might turn to the requirement of "industrial application" for assistance, we no longer live in the industrial age and should not expect a standard that was developed to protect industrial innovations to provide the right solution when applied to innovations in a different space. The value of advances in the categories of software under discussion lies not necessarily in their industrial application but in the other kinds of social benefits they provide. For better or worse, no appeal to existing standards such as "industrial application, "technical effect," "practical utility," or "useful, concrete, and tangible result" can enable us to avoid answering the underlying question whether software that performs functions falling squarely within the traditional definition of the "liberal arts" nevertheless should fall within the scope of patent law.

Answering such a question necessarily involves social value judgments and the resolution of public policy questions that are beyond the competence of patent examiners and judges to decide. Until an informed public debate is engaged and concluded on this question, therefore, the best that we can do is to work to develop pragmatic rules of decision which are consistent with existing law and which do as little damage as possible to the fundamental public policies underlying the patent system.

---

[58] *See* Richard Susskind, *The Future of Law* (Oxford 1996).

[59] Interestingly, this breakdown is evidenced not only in the functions performed by software, but in the way in which software is created. The act of programming a computer includes elements both of engineering and of literary authorship and has been described as an art as much as a science.

About "trivial" software patents: the IsNot case

Jan A. Bergstra[♣] Paul Klint[♠]

[♣] *Informatics Institute, University of Amsterdam*
and
Faculty of Philosophy, University of Utrecht
www.science.uva.nl/~janb

[♠] *Centrum voor Wiskunde en Informatica (CWI), Software Engineering Department*
and
Informatics Institute, University of Amsterdam
www.cwi.nl/~paulk

August 30, 2005

Abstract

So-called "trivial" software patents undermine the patenting system and are detrimental for innovation. In this paper we use a case-based approach to get a better understanding of this phenomenon. First, we establish a baseline for studying the relation between software development and intellectual property rights by formulating a life cycle for the patenting system as well as three variations of the software life cycle: the defensive patent-aware software life cycle that prevents patent infringements, the more offensive patent-based software life cycle that aims both at preventing infringements and at creating new patents, and the IPR-based software life cycle that considers all forms of protection of intellectual property rights including copyright and secrecy.

Next, we study an application for a software patent concerning the inequality operator and a granted European patent on memory management. We also briefly mention other examples of trivial patents. These examples serve to clarify the issues that arise when integrating patents in the software life cycle.

In an extensive discussion, we cover the difference between expression and idea, the role of patent claims, software patents versus computer implemented inventions, the role of prior art, implications of software patents for open source software, for education, for government-funded research, and for the current debate on the proposed EU patent directive. We conclude the discussion with the formulation of an "integrity axiom" for software patent authors and owners and sketch an agenda for software patent research.

We conclude that patents are too important to be left to lawyers and economists and that a complete reinterpretation of the patenting system from a software engineering perspective is necessary to understand all ramifications of software patents. We end with 12 explicit observations and recommendations.

1 Background

For many years, there have been concerns in the Unites States (US) about the possibilities to patent "trivial" software techniques and business methods. The patenting laws in the European Union (EU) have always been more restrictive than their US counterparts, but in the discussion about the proposed new EU directive about patenting computer implemented inventions (CII), or software patents for short, the level of triviality of a software patent has become a focal point in the debate: does a patent lay claims on techniques that are generally considered to be common knowledge or does the patent claim a real invention?

As part of a 3 year European Commission (EC) study[1] on the effects of software patents on innovation we are involved in a multi-disciplinary effort to understand the effects of software patents. These effects

[1] *Study of the effects of allowing patent claims for computer-implemented inventions*, a joint study by MERIT (University of

are studied from legal, economical, and computer science perspectives. The goal of the current paper is to study trivial software patents from a computer science perspective and to make a contribution to the discussion among experts from the three disciplines just mentioned. For economic effects we refer to [19, 11] and for legal aspects to [30, 5].

Software patenting is a relatively new topic for both authors, as it probably is for most software engineers and computer scientists. For completeness, we mention that both signed a petition to the European Parliament [31]. The second author has acted as speaker on a conference about the topic [13] (later adopted as point of view of the Royal Dutch Academy of Sciences) and has written a column about it [22]. Our professional interest in the topic stems from a long involvement in software engineering research ranging from study of the software life cycle [10], concepts of programming languages [9], theory, design and use of software components [6, 7, 8], generic language technology [12] and program analysis [21]. Both authors have cooperated in setting up the MSc curriculum in Software Engineering at the University of Amsterdam, now organized in cooperation with Mark van de Brand of the Hogeschool van Amsterdam and taught in cooperation with Hans van Vliet from the Vrije Universiteit in Amsterdam. Software patenting is therefore a major concern for us.

The plan of the paper is as follows. First, we start exploring how what seems to be a huge distance between the world of patents and the world of software engineering can be bridged. First we design in Section 2 a life cycle for the patenting process and next we make a connection between patents and software engineering by designing a patent-based software engineering life cycle (Section 3).

Given this conceptual framework, we study recent examples of software patents in order to get a better perspective on the implications for these software life cycles. In Section 4 we describe a recent patent application that might be a candidate for the predicate "trivial software patent". In Section 5 we present various views on this application. In Section 6 we briefly analyze a European patent on memory allocation and conclude that its novelty is strongly debatable. Next, we mention in Section 7 other trivial patents, both from the US and from Europe. A discussion (Section 8) and conclusions (Section 9) complete the paper.

2 The patent life cycle

It is important to describe the phases of the patenting process in such a manner that they become recognizable for the software engineer. We conjecture that the *Patent Life Cycle* shown in Figure 1 is a fair representation of this process. It consists of the following phases:

- An applicant *applies* for a patent.
- The applicant can decide to *withdraw* the application.
- The Patent Office can either *grant* or *reject* the application.
- The applicant can *appeal* against this decision and a reject decision may be changed into a grant decision.
- The applicant of a granted patent applicant becomes the holder of the patent.
- A granted patent may be *challenged* by another party. This may lead to revocation of the patent.
- The patent holder may act on *infringement* of its patent.
- The patent holder may *license* its patent to another party.
- The patent holder may *extend* its patent periodically.
- The patent *expires* after a maximal duration.

Maastricht, Netherlands), Centre of Intellectual Property Law CIER (University of Utrecht, Netherlands), Centrum voor Wiskunde en Informatica (Amsterdam, Netherlands), Telecommunication Engineering School at the Universidad Politécnica de Madrid (UPM), Spain and Centre for Research on Innovation and Internationalization (CESPRI) at Bocconi University, Milan, Italy.

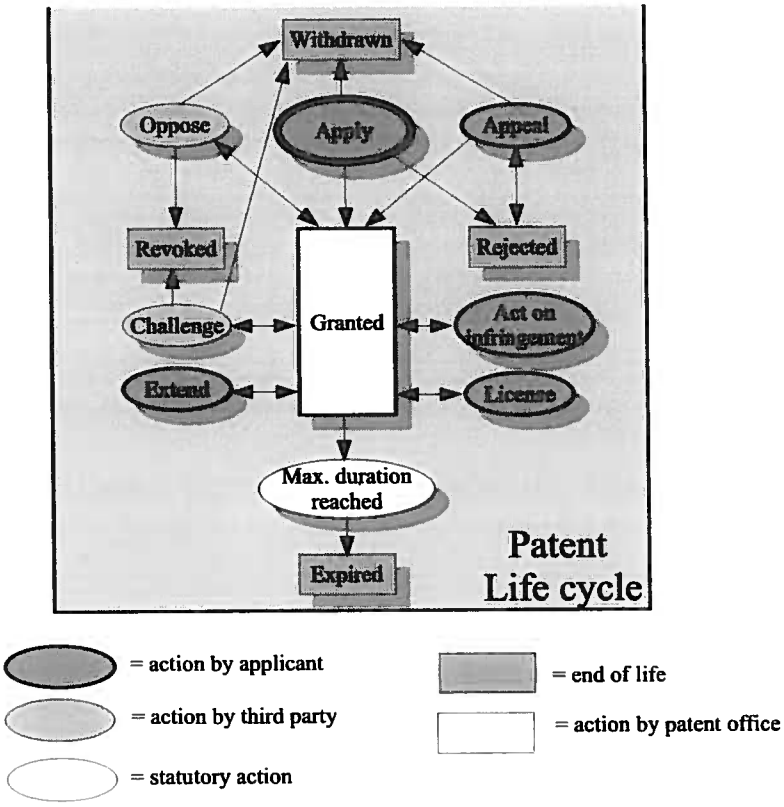

Figure 1: The patent life cycle: from filing to expiration

It is open for debate whether this abstraction of the patenting process can be used in the EU as well as in the US and Japan. However, since software developers have to be aware of potential patent infringements, independent of the source of the patent, such an abstraction of the patenting process is essential. This is relevant for developers of both commercial software and open source software.

The IsNot patent to be discussed later on in Section 4 is in the application phase, for all other patents mentioned in this paper we have explicitly indicated their status.

3 Baseline: an IPR-based software engineering life cycle

The next step is to make a connection between the patenting process—or rather Intellectual Property Rights (IPR) in general—and software engineering practices.

3.1 The software life cycle

In software engineering, the software life cycle is a frequently used manner of organizing the software development process. Figure 2 shows a strongly simplified version of the life cycle taken from a standard textbook [35]. It consists of the following phases:

- Requirements engineering: collect the requirements and expectations from the future owners and users of the system.
- Design: translate the requirements in a specification that describes the global architecture and the functionality of the system.

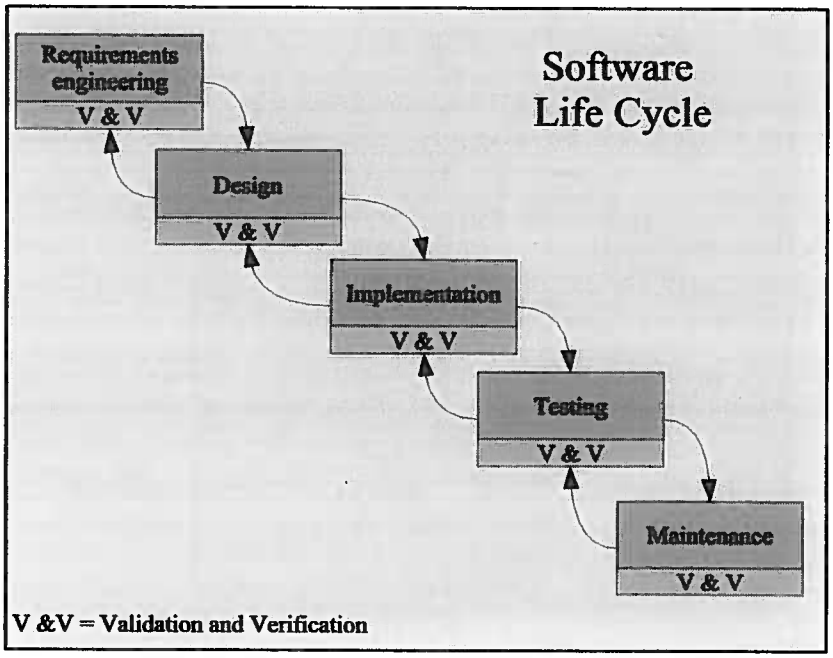

Figure 2: The software life cycle

- Implementation: build the system.This amount to transforming the design into software source code.
- Testing: test that the implemented system conforms to the specification.
- Maintenance: install, maintain and gradually improve the system.

It should be emphasized that the software life cycle covers design and construction of a software product as well as its use. Each phase contains a Validation and Verification (V&V) sub-phase in which the quality of the deliverables of that phases are controlled. Also note the backward arrows that make this into a real "cycle": it is possible to discover in later phases that decisions made in a previous phase have to be revised.

We will now proceed in three steps. First, a defensive Patent-aware Software Life Cycle is sketched that ensures that the software development organization does not infringe patents of third parties. Next, a more offensive Patent-based Software Life Cycle is described that also considers the options to file patent applications for knowledge that has been generated in each phase of the life cycle. Finally, the IPR-based Software Life Cycle extends the previous one to all IPR options: secrecy, copyrights and patents.

3.2 The patent-aware software life cyle

In Figure 3, we sketch a Patent-aware Software Life Cycle in which an extra sub-phase is added that performs patent validation. This generates immediately many unsolved questions. For each phase one may wonder:

- Is it possible to infringe patents in this phase?
- If so, how can one find such infringements?
- How can such infringements be resolved?

The Patent-aware Software Life Cycle is a defensive step that any commercial or open source software development process should adopt. Clearly the costs for software development will increase significantly.

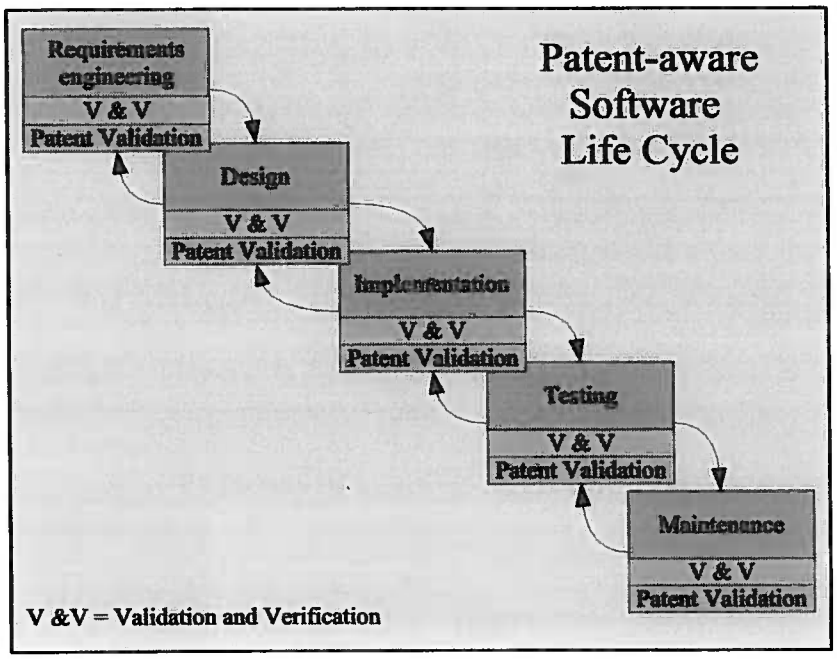

Figure 3: The defensive patent-aware software life cycle

3.3 The patent-based software life cycle

It is, however, possible to go one step further. In Figure 4 we sketch a Patent-based Software Life Cycle in which yet another sub-phase has been added that performs patent applications whenever possible. We conjecture that this strategy is only available to the software development organizations with the deepest pockets. For each phase now further questions apply, such as

- Does this phase generate patentable knowledge?
- Should we file a patent application for this knowledge?
- Are there other means to avoid that this knowledge generates an advantage for our competitors?

In many large software development organizations there exist "Chinese walls" between software developers and patent attorneys. This is not only the case for large commercial organizations but also for large open source projects like the Apache Foundation. The rationale being that the less software developers know about patents the stronger the position of the organization is in legal disputes. Implementation of the Patent-based Software Life Cycle may require similar measures. Of course, such measures completely defeat one of the primary goals of the patent system, i.e., knowledge dissemination.

3.4 The IPR-based software life cycle

The final step is the IPR-based Software Life Cycle sketched in Figure 5 that takes *all* aspects of IPR into account during software development. For each phase the questions now become:

- Does this phase violate copyrights of others? If so, remove those violations.
- Does this phase infringe patents? If so, negotiate a license with the patent holder or take technical measures to avoid the infringements.
- Does this phase generate valuable knowledge? If so, consider the following three options:

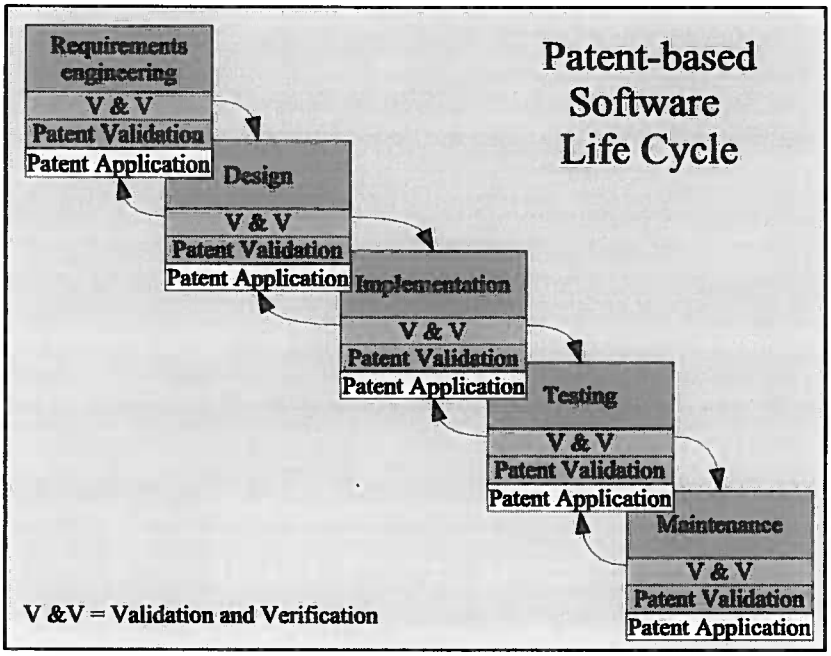

Figure 4: The offensive patent-based software life cycle

- Keep the knowledge secret and take appropriate legal or technical measures to achieve this.
- Establish copyrights on this knowledge.
- File patent applications for this knowledge.

These questions form a comprehensive description of the IPR policy one would expect of the biggest, multi-national, software development organizations.

3.5 Discussion

These extended software life cycles already raise many fundamental questions that are not easy to answer:

- Is it possible to use these extended software life cycles in such a way that they comply with the major patenting systems world wide?
- How can the software engineering knowledge that is hidden in the patent data bases made accessible for software engineers?
- Is it possible to a give an *operational* definition of a patent infringement that can be used by software engineers?
- For each of the phases of the software life cycle (requirements engineering, design, implementation, testing and maintenance) the following questions should be answered:
  - How is knowledge in this phase represented?
  - Where can prior art for this phase be found?
  - How can patent infringements in this phase be identified?
  - How can patent infringements in this phase be resolved?

  We expect that the answers to these questions will widely differ for each phase.

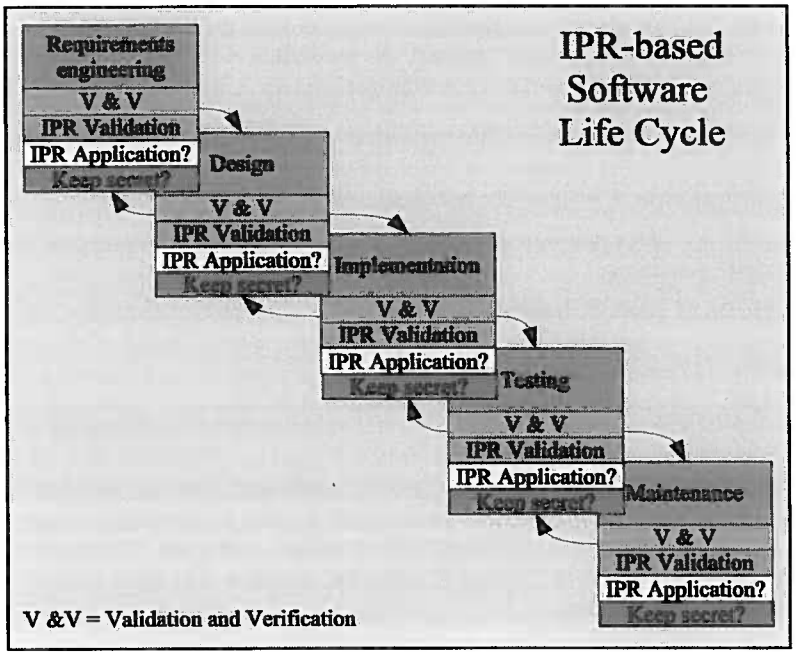

Figure 5: Considering all options: the IPR-based software life cycle

- What are the technical implications for software development when using these extended software life cycles?
- What are the economic implications of the extended software life cycles?

We will come back to these questions in the remainder of this paper and in Section 8.10 we will propose a research agenda.

We will now relate the high-level discussion in the previous sections to the daily practice of software patents by studying several examples.

4 The IsNot patent application

On May 14, 2003 the three Microsoft employees Paul A. Vick jr. (technical lead for Visual Basic), Cosica Corneliu Barsan (member of the Visual Basic compiler team), and Amanda K. Silver (program manager on the Visual Basic compiler team) filed United States Patent Application #437822 with the title "IS NOT OPERATOR". The abstract of the IsNot patent application (as we will call it) reads as follows:

> *A system, method and computer-readable medium support the use of a single operator that allows a comparison of two variables to determine if the two variables point to the same location in memory.*

The 8 page application consists of 24 claims followed by a description of the background of the invention, and detailed descriptions of illustrative embodiments. The first 5 claims of the application read as follows:

> *What is claimed:*
>
> *1. A system for determining if two operands point to different locations in memory, the system comprising: a compiler for receiving source code and generating executable code from the*

source code, the source code comprising an expression comprising an operator associated with a first operand and a second operand, the expression evaluating to true when the first operand and the second operand point to different memory locations.

2. The system of claim 1, wherein the compiler is a BASIC-derived programming language compiler.

3. The system of claim 1, wherein the operator is IsNot.

4. The system of claim 1, wherein the compiler comprises a scanner, a parser, an analyzer and an executable-generator.

5. The system of claim 4, wherein the source code comprises at least one statement, and the statement comprises a keyword representing the operator, the keyword recognized by the scanner.

The remaining 19 claims go into more details such as the parser determining that the operator is preceeded and followed by an operand, the fact that error messages are generated when the IsNot keyword or one of its operands are missing, the fact that executable code is generated, and so on and so forth.

The patent application describes that the invention can be used in exemplary computing environments ranging from PC, handhelds, servers, automatic teller machines, and more. The application also sketches in detail the hardware architecture of a typical PC using the invention. The application also explicitly states (in paragraph [0050]) the following:

It will be recognized that although in the examples, the operator is designated as "IsNoT", the invention is not so limited. Any suitable case sensitive or case insensitive tag for the operator is contemplated by the invention, such as, but not limited to "Is_Not", "isnot", "Isnot", "Is_Not", "is_not" and so on.[2]

5 Analysis of the IsNot patent application

5.1 IsNot is a trivial software patent

A lawyer or other non-specialist may be impressed by the clever invention described in the IsNot application, but each first-year computer science student will recognize what it is about: this is the inequality operator between pointer values as is known from many different programming languages ranging from the Branch Not Equal instruction `BNE` in PDP11 assembly language [14] to the not equal operator `.NE.` in Fortran [3] or the not equal operator `!=` in C [20], Java [17] or C# [16].

For a computer scientist, the *idea* of having a single operator for comparing two pointer values is common knowledge and the publications cited above constitute prior art.

For a computer scientist, granting this patent application will have devastating effects since it will cover a large majority of the software worldwide and will completely block any further software development or at least dramatically increase developments costs due to licensing.

5.2 IsNot is not a trivial software patent

We find it hard to believe that the highly skilled software developers at Microsoft (or their well-known colleagues at Microsoft Research) are unaware of the prior art mentioned above. It is also striking that prior art occurs in one of Microsoft's own products (the language C#). One is tempted to speculate about the intentions of the applicants and their sponsors with this particular patent application. Several possibilities come to mind:

- They think that the subject matter is new and this should be the default assumption. This raises the question whether there exists (or should exist) a form of "patent etiquette" that assumes that applicants truly consider their invention as new. According to Park [27], there is no explicit duty of

[2] It seems that the two occurrences of "Is_Not" are a typo in the application.

disclosing prior art in the European Patent Office, whereas the Patent Offices in the US and Japan require the applicant to disclose the closest prior art that he acknowledges when the patent application is filed. See Section 8.9 for a further discussion of this topic.

- They find that the matter of trivial patents and determining prior art need clarification and that filing a patent application is the fastest road to achieve this goal, independent of the likelihood of acceptance. This defines the future options for patenting relatively simple inventions. When rejected, the application builds up prior art and may be used to provide indemnity to clients against intellectual property claims.

What if our first analysis from a computer science perspective is too naive? Is it still possible to discover some form of innovation or hidden meaning in this application that merits its acceptance? We see the following possibilities for this:

- The patent application is about the *specific naming* of the comparison operator. This is suggested by the explicit phrase in the patent application we cited above: "*Any suitable case sensitive or case insensitive tag for the operator is contemplated by the invention, such as, but not limited to ...*". This would mean that the application is not about the idea of an inequality operator but about the specific form of that operator. In this way, the application would establish a form of copyright on the operator name "IsNot".
- The specific context of BASIC is the substance of the application. This also makes finding prior art hard.
- By patenting this specific operator in BASIC, alternative implementations of the language can be discouraged, or at least interoperability is hindered.
- The patent application is not concerned with the IsNot operator or the inequality operator at all. They just serve as a smoke screen to hide an idea in one of the 23 other claims. Is the patent about giving an error message when an operand of the IsNot operator is missing? Is this patent about BASIC-compilers using a certain compiler organization? As far as we can judge these claims describe common practices in compiler construction and language implementation and cannot be considered to be inventions. This does, however, not mean that it is easy to find prior art since most of the claims are very specific and may not occur in the literature. We invite the reader to investigate these claims and to challenge our analysis.
- The application has yet an other meaning, for instance, challenging the patenting system. In this case, we really congratulate the applicants for their brilliant contribution. Some implications are further discussed in the remainder of the paper.

5.3 Our opinion

Our opinion about the IsNot patent application can be summarized as follows:

- The IsNot patent application would, when granted, lead to a trivial patent and its inventive step does not differentiate itself from the manifest prior art given above. It is hard to understand why this application would be granted. When granted, this patent could indeed be very harmful for further development.
- In a similar fashion as each scientific publication needs a rationale, we miss a rationale for this patent application.
- It is undesirable that others would have the obligation to find prior art. Given the fact that US patent applications are required to disclose prior art, it is at least curious that this application gives none.
- It is unclear what an infringement of this patent (when granted) would mean. Is the design of a programming language that contains an inequality operator an infringement? Is every program that uses an inequality operator an infringement? Is the mere notion of an inequality test in any form an infringement?

- We don't see a convincing argument why a major company would need this patent, apart from tactical considerations where this patent may clearly play a role.
- Is this a typical patent application? It could be argued that this patent application is one of a kind, and that our analysis of it is thus irrelevant. Although we agree that this is one specific example of a trivial patent application, it is an application from a large firm with a large patent practice, and certainly sufficient resources to determine whether an "invention" is trivial, and to identify prior art, prior to submitting a patent application. So we believe that if IsNot may not be a typical patent application, it is certainly *potentially* typical.

6 Analysis of a European patent on memory allocation

We claim that a patent application is part of the patent life cycle (see Section 2) and is thus part of the open literature and should be publicly discussed and scrutinized for novelty and compliance with prior art. One may counter that the IsNot application may very well be rejected. From a European perspective, one may also counter that such an application would never be accepted by the European Patent Office (EPO). Therefore, we will also briefly analyze a patent granted by the EPO that we consider to be debatable.

On June 1, 1998, European Patent #817044 on "Memory allocation in a multithreaded environment" was granted to Sun Microsystems Inc. (US) with Nakhimovsky Gregory listed as inventor. The abstract reads:

> *A method of allocating memory in a multithreaded (parallel) computing environment in which threads running in parallel within a process are associated with one of a number of memory pools of a system of memory. The method includes the steps of establishing memory pools in the system memory, mapping each thread to one of the memory pools; and for each thread, dynamically allocating user memory blocks from the associated memory pool. The method allows any existing memory management malloc (memory allocation) package to be converted to a multithreaded version so that multithreaded processes are run with greater efficiency.*

The 8 page application consists of 21 claims followed by a description of the invention and preferred embodiments. The first 6 claims read as follows:

> *Claims of EP0817044*
>
> *1. A method of allocating memory in a multithreaded computing environment in which a plurality of threads run in parallel within a process, each thread having access to a system memory, the method comprising: establishing a plurality of memory pools in the system memory; mapping each thread to one said plurality of memory pools; and for each thread, dynamically allocating user memory blocks from the associated memory pool.*
>
> *2. The method of claim 1 wherein the step of dynamically allocating memory blocks includes designating the number of bytes in the block desired to allocate.*
>
> *3. The method of claim 1 further comprising the step of preventing simultaneous access to a memory pool by different threads.*
>
> *4. The method of claim 1 further comprising the step of establishing a memory pool for each thread comprises allocating a memory buffer of a preselected size.*
>
> *5. The method of claim 4 further comprising the step of dynamically increasing the size of the memory pool by allocating additional memory from the system memory in increments equal to the preselected size of the buffer memory.*
>
> *6. The method of claim 4 wherein the preselected size of the buffer is 64 Kbytes.*

The remaining 15 claims go into more details about the specific data structure to represent the memory pool and the memory blocks, and about the deallocation of blocks as well as their merger after deallocation.

Although the subject matter of this patent is not as astonishingly simple as that of the IsNot example, any computer scientist will see what this is about: memory allocation as it occurs in operating system

kernels and concurrent applications. A simple way to implement this is to have a single pool of memory blocks that can be claimed by one of the parallel processes (threads). However, to avoid corruption of the administration of the memory pool, access to the memory pool has to be strictly sequential. This is achieved by locking and unlocking the memory pool during each request. Since this locking introduces a time penalty, the idea formulated in this patent is to use a separate memory pool per process (thread).

In our opinion the idea to avoid the use of shared variables is trivial. We conjecture that this idea is not new and we think that there is a proof obligation on the part of the applicants to show how this patent improves upon earlier work (see Section 8.9).

This patent can have a major impact on the implementation of most, if not all, operating system kernels and its mere existence poses a threat to further development.

7 Other trivial patents

There are many examples of trivial software patents worldwide.[3] Examples are:

- US Patent 4648067: Footnote management for display and printing (IBM, 1987). This patent describes the handling of footnotes in a text processing system. This is a standard technique that has been used in every text processor since 1970.
- US Patent 5530794: Method and system for handling text that includes paragraph delimiters of differing formats (Microsoft,1996). This patent describes the conversion of text documents from Unix text files to MS Word format by inserting a carriage return character. Since the characters carriage return (CR) and line feed (LF) were invented, different operating systems have used them in different ways to end each line. This is a trivial technique that has been in use ever since.
- US Patent 5175857: System for sorting records having sorted strings each having a plurality of linked elements each element storing next record address (Toshiba, 1992). This patent describes sorting using linked lists. This is a standard technique found in every textbook.
- US Patent 6877000: Tool for converting SQL queries into portable ODBC (IBM, 2005): This patent describes how SQL queries can be translated into queries for the portable database interface ODBC. This obvious technique must be used by every database system that connects to ODBC.

Many other examples of trivial software patents are known.[4] From the fact that the above examples are all US Patents one might draw the conclusion that the US Patenting Office is more likely to issue trivial software patents. We think that this is not correct. The explanation is rather that the problem of trivial software patents has been in existence in the US for over 20 years and that the US patent databases have received more public scrutiny than, for instance, the European patent database. As an initial proof of this statement we have collected, in a very limited amount of time, the following European Patents that we consider to be trivial. Some trivial, but expired, patents are:

- European patent 10186: Apparatus for handling tagged pointers (IBM, 1980). This patent describes the addition of a tag bit to pointers in order to discriminate them from ordinary data. This is an old technique that has been used in various systems.
- European Patent 97818: Spelling verification method and typewriter embodying said method (IBM, 1984). This patent describes spell checking.
- European Patent 98959: Method for producing right margin justified text data in a text processing system (IBM, 1984).

[3] In this section we give examples of patents which we *suspect* to be trivial in nature and merit further study. A meticulous search for prior art is needed for each example. The fact that a patent is listed here does *not* imply a final judgment on our part of the patent's triviality or validity. The purpose of this section is merely to raise the awareness of potential trivial patents.

[4] See, for instance, `http://www.base.com/software-patents/examples.html`.

More recent examples are:[5]

- European Patent 698844: Tunnel icon (IBM, 1996). Describes a tunnel-like icon to which the user can drag files in order to encrypt or decrypt them. This patent is not particularly interesting or harmful but illustrates the level of detail and specificity the subject matter of a patent can have.
- European Patent 752695: Method and apparatus for simultaneously displaying graphics and video data on a computer display (Sun, 1997). This is a common technique for displaying information in a windowing system that has been in use for many years.
- European Patent 1043659: File signature check (Konami, 2000). This patent describes the use of checksums to detect whether files in a file system have been changed. This simple technique has already been used by many tools and is the obvious solution for this problem.
- European Patent 767940: Data pre-fetch for script-based multimedia systems (Intel, 2000). This patent aims at speeding up the execution of multimedia scripts running in a limited memory client. This is achieved by prefetching data references that occur in the script.
- European Patent 0195098: System for reproducing information in material objects at a point of sale location (Fpdc Inc, 1986).

  "This invention contemplates a system for reproducing information in material objects at a point of sale location wherein the information to be reproduced is provided at the point of sale location from a location remote with respect to the point of sale location, an owner authorization code is provided to the point of sale location in reponse to receiving a request code from the point of sale location requesting to reproducing predetermined information in a material object, and the predetermined information is reproduced in a material object at the point of sale location in response to receiving the owner authorization code."

  This patent (on downloading and burning CDs!) was recently overturned by the UK High Court [26]. As the EPO did not overturn it, it is unclear whether the High Court would have been able to overturn this if the Directive that enforced the EPO's status quo to be uniformly implemented across the EU were in place.

To conclude. we mention some recent patent applications:

- European Patent 1046117: Web browser graphics management (Philips, filed 1999). This patent application describes a prefetching mechanism for web browsers that has been floating around for many years, for instance in the Mozilla browser, where it is called link prefetching.
- European Patent 1014627: Constrained shortest path routing method (Lucent, filed 1999). This application describes an algorithm for shortest path calculation and seems to be a variation on Dijkstra's algorithm [15].

As already stated, the above examples constitute cases where we suspect that triviality and/or existing prior art make these patents or patent applications undesirable. Clearly, a public effort should be launched to scrutinize the European patent data base and look for trivial software patents.[6] Another public effort that is badly needed is to set up a searchable archive of prior art for software.

8 Discussion

8.1 Expression versus idea

The common view on copyright versus patenting is that copyright protects the *expression* of an idea, while a patent protects the *idea itself*. One idea can be expressed in many different ways. In other words, copyright

[5] See `http://swpat.ffii.org/patents/samples/index.en.html` for a more extensive collection of trivial European patents.

[6] The reader is invited to inspect the *Patent WIKI* database at `http://gauss.ffii.org/GaussFrontPage` and do some searching. It includes all patents issued by the European Patent Office. We can guarantee that there is some entertainment in this.

can protect one specific mystery novel, while a patent on the idea of a mystery novel itself will prevent anybody else to write mystery novels. The relevant US statute [34] reads as follows:

> *Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title.*

U.S. courts have summarized this principle by stating that patents do not apply to ideas themselves, but to "implementations" of ideas, intending a broader and more inclusive sense of "implementations" than is commonly given in the software development community. It is akin to the notion that copyright protects expressions of ideas rather than ideas themselves [25]. A similar definition can be found in the TRIPS treaty [36] that regulates trade-related aspects of intellectual property rights:

> *... patents shall be available for any inventions, whether products or processes, in all fields of technology, provided that they are new, involve an inventive step and are capable of industrial application.*

Now it happens to be the case that software engineering has its own ideology about the distinction between idea and expression of that idea. A *specification* of a software system describes the desired functionality of a system and defines *what* is required. The specification leaves open many options *how* the specified system will be built (the *implementation*[7] of the system).

For a software engineer, it is hard to understand why software patents (supposed to be about ideas, e.g., specification level) end up having detailed flowcharts that belong to the implementation level. This raises the issue how software engineering and the patenting system interfere.

Another observation is that the legal language in patents is about software. Now it also happens to be the case that the topic of software specification has a long history in computer science and, from a computer science perspective, the legal descriptions in patents can in no way be classified as such. In that sense patent texts are technically (but, of course, not legally) unacceptable for software engineers, which has implications for their utility as disclosure documents.

In our opinion, the patenting literature should take good notice of what is known about describing software systems. A crucial observation is here that the notion of formally describing an idea does not occur in the software engineering literature and it will be very hard to achieve this in patent texts. Another observation is that we think that it is unavoidable that patent texts will become machine processable documents that will form an integral part of software in a similar fashion as specification, documentation, test cases and the like form an integral part of a software system.

A final, and also crucial, observation is that the requirement that a patent should make a "technical contribution" is hard to reconcile with software that lives in the realm of logical structures. In this way, software patents have to be expressed in unnatural ways that lead to under-protection as well as over-protection of certain inventions.

This is eloquently described by Plotkin [28], he proposes a complete reinterpretation of the patenting system from a software perspective. His observation is that there are crucial differences between the invention, description and patenting of electromechanical devices as compared to software programs. His key observation is that for electromechanical devices apart from a functional design, deriving a physical structural design is hard and also essential for obtaining patent protection. In the case of software, the logical structures described by the source code are the end point of human invention: the step to their physical realization is fully automated. Plotkin's objectives are the following:

> *A methodology is proposed for determining how particular areas of law should apply to software. The methodology asks and answers four questions: (1) What is software?, (2) How does software differ from other creative works?, (3) How are such differences legally relevant?, and (4) How should the law treat software in light of such differences? Application of the first half*

[7] The word "implementation" is a multi-faced sword that may easily cause harm. In a legal sense, as in the previous paragraph, implementation denotes any method to go from idea towards its realization. Following that meaning, the specification of a software system is already an implementation of the idea for that system. In a computer science context, implementation always refers to the actual building and realization of a system in software.

of this methodology reveals that computer programs have the unique quality of being human-readable and computer-executable instructions that describe actions in purely logical terms. Application of the second half of this methodology to patent law and the First Amendment to the US Constitution reveals that software's unique features violate the law's assumptions, leading to results that are at odds with the underlying public policies in each case.

In later work [29] Plotkin proposes software patents in such a way that:

- *a software program be claimable solely in terms of its logical structure;*
- *a software program be patentable if:*
  - *the inventor provides a written description of the claimed logical structure;*
  - *the inventor provides a description that enables one of ordinary skill in the art to make and use the claimed program without undue experimentation;*
  - *the claimed logical structure has a practical utility;*
  - *the inventor conceives of the claimed logical structure;*
  - *the claimed logical structure is novel and nonobvious; and*
- *the scope of a software patent claim be limited to products and processes that embody the claimed logical structure.*

Other interesting proposals exist for reforming the patent system or for providing other forms of legal protection for software but they are not further discussed here. A concise summary of the history and current status of software patentability can be found in [18].

8.2 The role of patent claims

In [24] Lening and Cavicchi say about claims:

> *A claim is what an inventor is stating to be unique about the invention. The claims become the actual monopoly granted to the invention. Claims define the scope of protection granted to the invention.*

A claim can be *independent* (it stands by itself and is not dependent on another claim) or *dependent* (it makes express reference to a previous claim and depends on it). In the IsNot patent application, claim 1 is an independent claim while claim 2 is a dependent claim. A patent may also contain descriptions of preferred embodiments but they just serve as illustration and may at most be used to interpret the claims. A patent may contain both independent and dependent claims and the question arises what an infringement of a patent means *exactly*[8]

- If one (all) independent claims is (are) violated?
- If one (all) dependent claims is (are) violated?

The precise procedure for interpreting the claims in a patent seems to be an "art" and is a matter of debate among lawyers [4]. This is unsatisfactory from a software engineering perspective.

The status of claims needs also further clarification in the light of the "expression versus idea" discussion given earlier in Section 8.1. The question being: *what is an infringement?*. From the perspective of the software engineering life cycle (Section 3) the following questions need clarification:

- Is infringement possible during requirements engineering?
- Is infringement possible during design?
- Is infringement possible during implementation?

[8]The same questions can be asked when searching for prior art related to patent applications.

- Is infringement possible during testing?
- Is infringement possible during maintenance?

To be on the safe side, we have assumed in our patent-based software life cycle that the answer to all these questions is "yes". However, the nature of such infringements will be completely different, both in their description, appearance, and discovery.

During requirements engineering and design, only the intended behavior of the system is available. It is for instance, impossible to observe a running version of the software. Infringements can only be discovered by a deep semantic comparison between patent text and design documents.

During implementation, the desired behavior is coded as software program. Now it becomes possible to observe the behavior of the software by executing it on a computer. It also becomes possible to perform more syntactic comparisons between patent text and program text.

Software is both human-readable and computer executable, and this makes it unique among patentable artefacts.

8.3 Software patent versus computer implemented invention

We have, so far, spoken about "software patents". However, the proposed EU patent directive speaks about "computer implemented inventions" (CII) rather than software patent.

It may be maintained that software patents do not exist and that CII is the right phrase to use. We completely agree that the notion of a computer implemented invention is a meaningful one and that such inventions may be in need of patenting. In such cases computer programs may be used as an implementation strategy but a pure hardware implementation may be conceivable as well. In our opinion, all patents discussed in Section 7 are software patents in a more generic sense. The patent is about how to achieve something by means of running computer executable programs (software), or even on methods for writing such programs or designing programming languages. None of these inventions makes any sense outside the realm of programmed computers, and these inventions are about how something may be achieved given that computer programs will be used. *A software patent concerns an invention about a software-based computer implementation, while a computer implemented invention is about an invention that may be implemented in software.*

We cannot imagine that the IsNot patent application could be classified as a computer implemented invention which may admit a pure hardware embodiment, since this would amount to a single not gate. As a consequence the mere need to grant patents for clear cases of computer implemented inventions (e.g., the design of novel control software/hardware for an airbag) should *not* be taken as an argument that pure software patents do *not* exist. The software industry will soon be in a need to deal with a massive number of "true software patents". A careful consideration of the rules of that game from a software engineering perspective is necessary to grasp the effects of the introduction of patent regulations that will generate an abundance of such patents.

8.4 The role of "prior art"

Prior art is defined as the body of prior knowledge relating to the claimed invention, including prior use, publications and patent disclosures [24]. During the patent life cycle (Section 3), prior art plays a role at different moments:

- When an application is rejected, the applicant can dispute prior art that is used in the motivation of the rejection.
- When the patent is challenged, the challenger has to produce prior art that invalidates one or more of the claims of the patent.
- When the patent holder acts on an alleged infringement of its patent by a third party, he must show that the third party uses results or methods that are claimed by the patent (one could call this "posterior art").

In the patent application it is usually indicated which previous patents are used or extended. As already discussed in Section 5.2, only the European Patent Office does not require to mention prior art that is known to the applicant.

We conjecture that in all the three cases mentioned above, the determination of prior art is identical, whether this is true prior art or posterior art as defined above.

A patent may describe a technique that computer scientists consider to be trivial. Nonetheless, it may turn out to be very hard to find prior art for it. Well-known techniques cannot be published in a scientific publication for the simple reason that they are already well-known and do not constitute a new research result. These well-known techniques may be used in the source code of many software systems, but this does not count as "publication" and cannot be used to illustrate prior art. At the same time, it may also be the case that they are not covered by any patent and someone can just file a patent application for this well-known technique.

Ullman [33] is among the most cited computer science researchers world-wide and he describes eloquently the difficulty to find prior art for a patent application about matrix triangularization that was later used in an unsuccessful attempt to bring suit to the large spreadsheet manufacturers.

In disciplines like chemistry and biology the patent literature forms the actual documentation of inventions. For software the unique situation exists that there is another powerful information source that plays no role in the patent process: the source code itself. This is a major handicap when searching for prior art. There is evidence that cross-citation between the patenting literature and the computer science literature is nearly absent [1]. Compared to software patents, business patents seem to contain relatively more references to the non-patent literature [2]. Nonetheless, the world of software and the world of patents seem mostly disjoint. From this follows that computer scientists are currently not well-aware of the patent literature.

We may conclude that it is urgent to find new ways to establish prior art. One way is the creation of public web sites that solicit and award proofs of prior art. It seems reasonable to include procedures in the patenting system where the public can submit prior art against patent applications.

Another way for establishing prior art is the patent system itself. Suppose the IsNot application is rejected. This fact can have a very positive impact: all the claims in the application are considered to be un-patentable and this blocks future patents on the issues stated in the rejected claims. In this way, a rejected patent application contributes to building up prior art. It is conceivable that major companies follow this strategy in order to prevent patent applications by competitors or to provide indemnity to clients against intellectual property claims.

8.5 Implications for Open Source Software

There has been active opposition from the Open Source Software (OSS) community against the emergence of a system for software patenting. As discussed earlier in Section 8.2, the assumption that open source software products allow inspection at a syntactic level does not imply a greater risk for infringement detection. To establish that this risk would be higher requires a very clear understanding of what constitutes an infringement of a software patent and how to establish such an infringement. As discussed earlier, exactly this understanding is missing. On the contrary, OSS producing companies or individuals may often afford to distribute quite vague functional specifications using the fact that their user community is willing to take some risks and to accept some trial and error, whereas producers of closed software components need to specify in meticulous detail what is to be expected from their products and this may even give better clues for those who search for potential patent infringements. We see therefore no reason why the authors of open source should be more (or less) worried about the potential implications of software patenting for their business than the authors of closed software, from the perspective of establishing infringement.

It is true, however, that the distributed development model of open source rests upon a legal infrastructure–open source licenses–that assume that individual authors own, and thus have the right to "give away", whatever they write. While this works in the copyright system, it is incompatible with patents, since individual developers can no longer assume that they own what they write, and can thus never know whether they have the right to "give it away" through open source licenses. This is a subject of further research in the course of the on-going study.

Many commercial manufacturers are now disclosing sources under limited licensing schemes while making use of substantial copyright protection. The variation of licensing schemes has much impact on the economic models used and only some licenses lead to the much debated cost reduction that many people consider typical for open source software. Source pricing and source disclosure are independent matters: open source software may even be quite expensive in some cases. If that were not the case (in principle) the whole patenting system should be considered irrelevant as such because it only protects users and producers of disclosed information.

8.6 Implications for education

Patent law requires that a patent should be non-obvious to a "person skilled in the art". As already observed by Ullman [33], it is unclear what the background of such a person should be: ranging from a self-educated programmer, via a bachelor or master in computer science or software engineering, to a professional researcher in these areas. If we consider the Software Engineering Body of Knowledge (SWEBOK [32]) as approved by IEEE, we are pretty sure that a person with that knowledge is unable to read or interpret software patents let alone determine potential infringements. The patent-aware software engineering life cycle (Section 3) also requires an increased level of awareness of software patents among software engineers as well as the skills to turn this awareness into deeds.

It is clear that the current education of software engineers and the future requirements imposed by a patenting system including software patents will be dramatic. As far as we aware, there is no curriculum worldwide that is prepared for this. Governments should invest in the development of such curricula and in major retraining of professional software engineers.

8.7 Implications for government-funded research

Software is developed in many research projects that are being funded by national governments. Most of these project follow a traditional software life cycle that ignores patents. In order to avoid that governments become vulnerable for extensive infringement claims, they should require that these projects switch to at least a patent-aware software life cycle. This will require extensive additional funding for these projects.

8.8 Implications for the current debate on the software patent directive

The introduction of software patents in any form immediately raises the following questions:

a What constitutes prior art, and what is the status of existing programs.

b How to avoid trivial patents.

c How to design a patent-based software engineering life cycle.

d How to design a patent aware life cycle (less crucial but economically vital).

In the current debate in the EU we get the impression that acceptance of the draft directive "on the patentability of computer-implemented inventions" (software patent directive), as it stands would lead to de-facto software patents (in spite of the CII jargon) without the prerequisite clarification concerning the issues listed above, thus creating unpredictable legal risks for many parties involved.

Amending the directive to such extent that there is no legal basis left for the protection of any software components deprives manufacturing companies from legal means available to them now, and thereby introduces additional risks just as well. This seems to lead to the position that neither the directive nor a version of it that cuts out any IPR protection for software components (or against infringing software components) is a step forward.

The current proposals seem to focus on software/hardware component specifications that constitute a vital part of CII's. The functional specification of a unit is given (as part of the proposed CII architecture) and then an infringement may result by producing a software component that meets the specification even if the manufacturer has shown the ability to implement the specification by means of the description of a piece of hardware. Thus some branches of industry propose (understandably) a capability to provide

this form of protection. Unfortunately, the resulting scope of IPR and infringement protection has been insufficiently demarcated.

The modularization paradox Some assume that by requiring that embodiments of a patent have effects that depend on laws of nature (though excluding software as such) conceptual problems can be solved. This cannot be excluded *per se* though it may get paradoxical as follows:

- One may describe an invention as an application in technology rather than dealing with laws of nature.
- One may consider computer programs as software and one may also consider software as belonging to technology.
- Now consider computer programs P and Q where Q provides a context within which P may work. On the one hand $P + Q$ cannot be patented as it is 'software as such', on the other hand P may be patented because of its role it may play in the context of Q (which is a technical context given the above assumptions).

Taking this observation to the extreme: in a context where software as such cannot be patented and technical effects are required, one may be tempted to split a software invention into claimed components and stated components where the stated components are part of the justification of the claimed components. Interestingly this introduces a tendency to trivialize a patent description. More importantly, however, the whole state of affairs with P and $P+Q$ is conceptually inconsistent. Therefore the dogma's that software as such cannot be patented *and* technical effects are required make sense only in a setting where one assumes beforehand that a collection of software components never represents a part of technology.

How to move ahead? Given the fact that world-wide a large number of de-facto software patents exist (even if a jargon is used that suggests these patents to be of another nature) it is already now important for the EU to initiate substantial research and development for the clarification of the questions **a–d** mentioned above. On the basis of such work technology can be developed that takes into account all existing patent databases. In successive stages limited possibilities for the protection of

- software/hardware components specifications,
- software component implementations,
- software architectures,
- software processes (software engineering methods)

may be developed.

By doing this kind of work the EU will possibly lead the way in sophisticated use of software patent databases while at the same time preparing for patenting regulations that really work. In terms of software engineering these regulations themselves are just some form of standard concerning the software process. It is clear that such a standard should only be enforced if it has substantial informal backing and if the technology supporting it is sufficiently sophisticated.

We expect that in the long run software patents will indeed emerge and that this will lead to a wealth of supporting technology. What is at stake here, is the risk that the EU misses the opportunity to leap ahead by developing sophisticated legislation in which software is a first class citizen and *also* misses the opportunity to develop the technology for supporting such legislation.

That leads us to this position: a sophisticated patenting system for (categories of) computer software will enhance software technology in the EU, provided that the considerations given here are addressed on a reasonably grand and effective scale. Introduction of a software patenting system without these pre-requisites in place will have disappointing effects. The current opposition against the EC proposals is a manifestation of these disappointing effects.

8.9 Integrity axiom for software patent authors and owners

We recommend to add the following integrity axiom to the assumptions about software patents: *every patent which is either live or in the application phase expresses the views held by its authors and owners in the following way*:

- The described invention did not conflict with prior art (in the most general sense of this expression) when it came into existence and by definition has been so ever since. In addition, the patent is non-trivial at that same moment in time.
- The patent authors rightly claim as professional software engineers the IPR for said invention.
- This IPR entitles them to economic revenues in an enforcible way.
- If the patent is owned by an organization that employs one or more of its authors, the relevant management layers of this organization share the views stated above.

This axiom is non-obvious because filing a patent application is an action by some agent and the axiom is about the mental state of that agent.

One might drop the integrity axiom in which case software patenting becomes some form of gaming not primarily based on the meaning of the patents but rather on their tactical and dynamic properties. For instance, a company might file a sequence of trivial patents just to exhaust the capacity of an economic opponent to effectively complain about these applications in order to arrive at a stage where IPR can be claimed even if it is not justified in real terms. But if the only way to get something out of patents would be along these lines we tend to agree with Knuth [23] that the whole enterprise is flawed. The integrity axiom excludes tactical patenting which is not based on reliable facts. This is very similar to scientific publications which are also supposed to adequately represent author's views.

8.10 A research agenda for software patent research

Taking software patents seriously means designing patenting systems and studying their implications. Here are some suggestions for a research agenda.

Current status of software patenting regimes One should take into account at least what happens in the USA, the EU, Japan, India and probably more. The Gauss database mentioned earlier is an example of such work. Here one finds the systematic investigation into the non-triviality and prior art violations of existing patents. We can imagine that a patent monitor is developed which enables the public to systematically submit their opinions about existing patents. In addition various forms of text-mining and cluster analysis can be employed to unlock the knowledge in the patent databases.

Revision proposals concerning the various regimes Several proposals have already been made for revising the various regimes. These should be studied and compared in detail.

Designing possible software patenting regimes There is no reason to believe that one unique software patenting regime can be designed, assuming that one exists at all. Thus many different regimes should be investigated. For each regime a set of questions has to be settled: what constitutes prior art, what is an infringement, how to define the particular 'patent speak' and its semantics, definition of the appropriate life cycles, and so on.

An important step might be to develop a collection op hypothetical software patents, i.e., rewrites and perhaps simplifications of the software development history in which known developments are ordered in such a way that some steps can convincingly be patented. The historical development of computer software might even be simulated in a game-like fashion in order to study the impact that some patents (had they existed) might have had.

Collection of prior art A crucial element in any patenting regime is the role of prior art. We propose to investigate the possibilities for

- formalizing prior art, i.e., all relevant knowledge about software;
- formalizing the claims in patents;
- comparing formalized patent claims with formalized prior art;
- automated searches for patent infringements in existing software, given formalized patent claims.

We believe that this research agenda can contribute to a revision of the patent system and may even lead to a form of software patents that behave as intended: disseminate the knowledge about inventions and give rewards to true inventors.

9 Conclusions

Our main conclusion is that patents are too important to be left to lawyers and economists and that the only way to fully understand the ramifications of software patents on existing software engineering is to completely reinterpret the patenting system from a software engineering perspective. This will require extensive study and will also create competitive advantages for the EU. Now, hastily, accepting the proposed directive on the patentability of computer-implemented inventions" (software patent directive) will have adverse effects. More specifically, we have shown the following:

1. Software is both human-readable and computer executable and this makes it unique among patentable artefacts. The requirement that a patentable invention should make "a technical contribution" leads to unnatural descriptions of software inventions and to inadequate claims.
2. Rejected trivial software patents are a tool for establishing prior art.
3. The European Patent Office should require that patent applications mention all prior art (not only from the patent literature but especially from sources outside the patent literature) that is known to the applicants. In practice, disclosure or even awareness of prior art is avoided for legal reasons (see the discussion on "Chinese walls" in Section 3). This is an undesirable situation since it undermines one of the primary roles of the patenting system: acting as a knowledge dissemination mechanism.
4. A public effort should be launched to scrutinize the European patent data base and look for trivial software patents. Such a public validation phase should become part of the patent application procedure.
5. The sources on which prior art searches are based should be extended in the case of software patents; in particular web-sites, mailing lists, and software source code should be permitted as sources of prior art.
6. There is a need for a patent life cycle that can be used to better understand the patenting process; in this paper we propose such a life cycle. Since software developers work worldwide, the patent life cycle should abstract from specific patenting regimes (EU, US, Japan).
7. We propose software life cycles that are patent-aware (defensive), patent-based (offensive), and IPR-based (includes copyright, patents and secrecy). They are needed to reconcile software engineering practices with the patenting system.
8. Adopting any patent-related software life cycle increases the costs of software development.
9. The fact that patenting of certain computer implemented inventions might be reasonable should be considered independently from the implications of pure software patents. New forms for the protection of software inventions should be studied.

10. Governments should make major investments in designing patent-based curricula for software engineering and computer science as well as in retraining programs for professional software engineers.

11. Governments should require that all software development that takes places in projects they fund follow the patent-aware software life cycle. Otherwise, governments may become vulnerable for infringement claims.

12. The EU should launch collaborative efforts to collect and categorize prior art in software engineering. This will lead to a defense against software patents from outside the EU and it will also advance the level of knowledge and technology to effectively handle patent information.

We cannot resist to conclude this paper with a quote from the world-famous Donald Knuth, professor emeritus from Stanford University, in a letter to the US Patent Office [23], since it clearly summarizes the opposition against the proposed software patent directive in the EU (although we do not draw the conclusion that software patents are a bad idea under all circumstances):

> *The basic algorithmic ideas that people are now rushing to patent are so fundamental, the result threatens to be like what would happen if we allowed authors to have patents on individual words and concepts. Novelists or journalists would be unable to write stories unless their publishers had permission from the owners of the words. Algorithms are exactly as basic to software as words are to writers, because they are the fundamental building blocks needed to make interesting products. What would happen if individual lawyers could patent their methods of defense, or if Supreme Court justices could patent their precedents?*

Acknowledgments

We thank our partners in the *Study of the effects of allowing patent claims for computer-implemented inventions* for their insights and help. Reinier Bakels was very helpful in answering our questions about legal matters, Bronwyn Hall pointed to relevant references, and Rishab Ghosh kept pressuring us to include more European patent examples. All three reviewed drafts of this paper.

Erik Josefsson was helpful in pointing us to interesting EU patent applications and for setting up `http://gauss.ffii.org/GaussFrontPage` as a useful tool for patent research. Dirk-Willem van Gulik draw our attention to the importance of Chinese walls between software developers and patent attorneys. Paul E. Merrell pointed to inaccuracies in our draft descriptions of what is patentable and his suggestions greatly helped to clarify this. Jan van Eijck helped to increase our insight by challenging our assumptions about patents and Jo Lahaye was a stimulating discussion partner on this topic.

References

[1] G. Aharonian. Patent examination system is intellectually corrupt. `http://www.bustpatents.com/corrupt.htm`, May 2000.

[2] J.R. Allison and E. H. Tiller. Internet business method patents. In W. M. Cohen and S. A. Merrill, editors, *Patents in the Knowledge-Based Economy*, pages 259–284. National Research Council, Washington, National Academies Press, 2003.

[3] ANSI. `http://www.fortran.com/F77_std/f77_std.html`, 1977. ANSI Standard X3.9-1978 and ISO 1539-1980.

[4] R. Bakels, 2005. Private Communication.

[5] R. Bakels and P.B. Hugenholtz. The patentability of computer programmes: Discussion of European level legislation in the field of patents for software. Technical report, European Parliament, 2002.

[6] J. A. Bergstra, J. Heering, and P. Klint, editors. *Algebraic Specification*. ACM Press/Addison-Wesley, 1989.

[7] J.A. Bergstra, J. Heering, and P. Klint. Module algebra. *Journal of the ACM*, 37(2):335–372, 1990.

[8] J.A. Bergstra and P. Klint. The discrete time ToolBus – a software coordination architecture. *Science of Computer Programming*, 31(2-3):205–229, July 1998.

[9] J.A. Bergstra and M.E. Loots. Program algebra for sequential code. *Journal of logic and algebraic programming*, 51(2):125–156, 2002.

[10] J.A. Bergstra and S.F.M. van Vlijmen. *Theoretische software engineering, kenmerken-faseringen-classificaties*, volume XXVIII of *Questiones Infinitae*. Zeno instituut voor Filosofie (Leiden-Utrecht), 1998. (In Dutch).

[11] J. Bessen and R.M. Hunt. An empirical look at software patents. Economics Research Working Paper 03-17/R, Philadelphia Federal Reserve Bank, March 2004.

[12] M.G.J. van den Brand, A. van Deursen, J. Heering, H.A. de Jong, M. de Jonge, T. Kuipers, P. Klint, L. Moonen, P.A. Olivier, J. Scheerder, J.J. Vinju, E. Visser, and J. Visser. The ASF+SDF Meta-Environment: a Component-Based Language Development Environment. In R. Wilhelm, editor, *Compiler Construction (CC '01)*, volume 2027 of *Lecture Notes in Computer Science*, pages 365–370. Springer-Verlag, 2001.

[13] Software patents: the choice is yours. `http://www.softwarepatenten.be/conferenties/september03`, September 17, 2003. Brussels.

[14] Digital Equipment Corporation. *Processor Handbook PDP11/45*, 1974.

[15] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[16] ECMA International. *C# Language Specification*, 2nd edition, December 2002. ECMA-334, `http://www.ecma-international.org/publications/standards/Ecma-334.htm`.

[17] J. Gosling, B. Joy, and G. Steele. *The Java Language Specification*. Addison-Wesley, 1996.

[18] J. Halbersztadt. Remarks on the patentability of computer software – History, Status, Developments. `swpat.ffii.org/events/2001/linuxtag/jh/swplxtg017jh.en.pdf`, April 2001.

[19] B. H. Hall. Innovation and market value. In R. Barrell, G. Mason, and M. O'Mahoney, editors, *Productivity, Innovation and Economic Performance*, pages 177–198. Cambridge University Press, 2000.

[20] B.W. Kernighan and D.M. Ritchie. *The C Programming Language*. Prentice-Hall, 1978.

[21] P. Klint. How understanding and restructuring differ from compiling—a rewriting perspective. In *Proceedings of the 11th International Workshop on Program Comprehension (IWPC03)*, pages 2–12. IEEE Computer Society, 2003.

[22] P. Klint. Een patentoplossing? Nee, dank U! *I/O Informaticaonderzoek*, 1(2):3, September 2004. (In Dutch), `http://www.informaticaplatform.nl/images/uploaded/magazine_2004_12_IO2t%otaal.pdf`.

[23] D. Knuth. Letter to the US patent office. `http://lpf.ai.mit.edu/Patents/knuth-to-pto.txt`, September 2003.

[24] C. Lening and J.R. Cavicchi. Patent searching glossary. Technical report, Franklin Pierce Law Centre, 2003. `http://ipmall.info/hosted_resources/patent_searching_glossary.pdf`.

[25] P.E. Merrell, 2005. Private Communication.

[26] M. Murphy. Getty and corbis win image patent dispute. Seattle Post-Intelligencer http://seattlepi.nwsource.com/business/227728_gettycorbis09.html, 2005.

[27] J. Park. Evolution of industry knowledge in the public domain: Prior art searching for software patents. *SCRIPT-ed*, 2(1), 2005. http://www.law.ed.ac.uk/ahrb/script-ed/vol2-1/park.asp.

[28] R. Plotkin. From idea to action: toward a unified theory of software and the law. *International Review of Law, Computers & Technology*, 17(3), November 2003.

[29] R. Plotkin. Computer programming and the automation of invention: a case for software patent reform. Working Paper Series, Public Law & Legal Theory Working Paper 04-16, Boston University School of Law, 2004.

[30] P. Samuelson. Should program algorithms be patented? *Communications of the ACM*, 33(8):23–27, 1990.

[31] 30 European Computer Scientists. Petition to the european parliament on the proposal for a directive on the patentability of computer-implemented inventions. *CEPIS UPGRADE The European Journal for the Information Professional*, IV(3):24–25, June 2003. http://www.upgrade-cepis.org/issues/2003/3/upgrade-vIV-3.html.

[32] Software engineering body of knowledge (SWEBOK). http://www.swebok.org, 2004.

[33] J.D. Ullman. Ordinary skill in the art. http://www-db.stanford.edu/~ullman/pub/focs00.html, November 2000.

[34] United States Code (USC). Title 35, Section 101: Inventions patentable. http://caselaw.lp.findlaw.com/scripts/ts_search.pl?title=35&sec=101, January 22 2002.

[35] H. van Vliet. *Software Engineering: Principles and Practice*. Wiley, second edition, 2000.

[36] World Trade Organization (WTO). Trips: Agreement on trade-related aspects of intellectual propery rights, Section 5, Article 27: Patentable Subject Matter. http://www.wto.org/english/tratop_e/trips_e/t_agm3c_e.htm.

Paradigm Shift in European Intellectual Property Law? From Microsoft to Linux

Lex Electronica, vol. 10, no 3 (special issue 10th Anniversary), Fall 2005
http://www.lex-electronica.org/

Lucie GUIBAULT*

1. Open Source and Copyright Law 3
1.1 Freedom to use 4
1.2 Freedom to reproduce 8
1.3 Freedom to modify 9
1.4 Freedom to (re)distribute 12
1.5 Royalty free distribution 16
2. Open Source and Patent Law 18
2.1. Open source and patented software 20
2.2 Open source patenting strategy 24
2.3 Open source licensing strategy 26
3. Enforcement of Open Source Licences 28
4. Conclusion 31

Open source or free software[1] is actually as old as the software industry, but its use is becoming more and more widespread among businesses, governments, and the public at large. Open source software licences are based on two fundamental principles: the possibility for users to use the software for any purpose and to modify and redistribute it without prior authorisation from the initial developer. Some open source software licences, like the General Public Licence (GPL), also impose a corollary obligation on the licensee: to make the source code available to other developers.[2] The idea behind this form of licensing is that when programmers can read, redistribute, and modify the source code for a piece of software, the software evolves.[3] Perhaps more than any other type of software, open source software is, as a result of its characteristic licensing scheme, the engine of collaborative creation. The very fact that the software may be freely used, modified and redistributed encourages subsequent developers to make their own contribution to an existing piece of software, by correcting errors, or by enhancing the software's capabilities and efficiency. Open source software may be developed in a closed setting, but it may also consist of a patchwork of different contributions originating successively from a number of unsupervised and unrelated developers, who are often scattered across different locations in the world. The modifications brought to the initial software can then either be distributed as a separate programme or be integrated into the original software.

Within a few years, the 'open' method of development and distribution of computer programs has imposed itself as a powerful social ideology. The philosophy behind open source licensing has also inspired the development of numerous other 'open' licences and 'open' projects, where the principles of open source are applied in the fields of music, media,

---

* (LL.M. Montréal en LL.D. Amsterdam) Senior researcher at the Institute for Information Law (IViR) of the University of Amsterdam (L.Guibault@uva.nl). This paper is adapted from O. van Daalen and L. Guibault, *Unravelling the Myth around Open Source Licences – An Analysis from a Dutch and European Perspective*, The Hague, T.M.C. Asser Press, forthcoming end 2005 or beginning 2006.

1 In this text, the expression "open source" software licences encompasses "free" software. For a distinction between the two movements, see: Richard Stallman, 'The Free Software Definition', at http://www.gnu.org/philosophy/freesw.html.

2 Free Software Foundation Europe, http://fsfeurope.org/documents/freesoftware.html.

3 Open Source Initiative, http://www.ossl.nl/opensource.org/

encyclopaedia and science. The mechanism for achieving this goal is through a standardized licensing infrastructure. The open source movement is so powerful in fact that even the software giant Microsoft felt the pressure to offer open and royalty-free documentation and licences for the Microsoft Office 2003 XML Reference Schemas, which provide developers and representatives of business and government a standard way to store and exchange data stored in documents.[4] Microsoft's release of the Office 2003 XML Reference Schemas does not qualify as 'free' or 'open source' software, for the accompanying licence does not grant the user the required freedom to use, reproduce, modify and redistribute the software. Nevertheless, Microsoft's gesture does give an indication of the increasing pressure of disclosing software standards within the community of software developers. Other important 'proprietary' software companies are slowly following Microsoft's footsteps and disclosing certain components of their products to the open source community.[5]

The use of open source software licences has given rise to new, viable, and attractive business models for the distribution of software products. In view of its commercial potential, established companies are investing important capital and labour resources in the development of open source operating systems and applications. Open-source licences cover thousands of projects, including the heart of the Linux operating system, the Firefox Web browser, the Apache server software collection and soon, Sun Microsystems' Solaris version of Unix. Open source software owes its attractiveness to the very principles put forward by its proponents: software users and developers savour the political freedom granted under the licence to use and modify the software as they wish.[6] The principles underlying the open content movement have been embraced by a large and varied public worldwide, including in the Netherlands, ranging from governments, to businesses, individual users and institutions. To some extent, however, the open source ideology may be victim of its own success, for the number of different open source licences has dramatically increased over the past couple of years, giving to rise to compatibility and transparency problems.

The increasing popularity of open source software licences could be the indication of a gradual paradigm-shift in the (use of the) law: from a strict protection regime for computer software following the Microsoft model, the trend moves towards an open and permissive form of protection, following the Linux model. While the protection computer software under copyright and patent law is premised on the idea that the grant of exclusive rights actually encourages creativity and innovation, the open source movement has emerged precisely in reaction to the perceived failure of traditional intellectual property law to do just that. In the opinion of the adherents to the open source software movement, protecting computer software under copyright and patent law leads to a contrary result primarily because of the constant expansion of intellectual property law and of the way rights holders license their rights to users and subsequent developers. Open source software licences offer an alternative to the traditional licensing model with the view of enabling access to computer information and of encouraging further software development. The subject of the impact of the open source software movement on law and practice appears particularly well suited for the Lex Electronica's tenth anniversary special issue in which authors are invited to explore various trends with regard to the impact of new technologies on the Law. This paper therefore presents the main elements of the most commonly used open source software licences, focusing on the General Public Licence (GPL), the Berkeley Software Distribution (BSD) and the Mozilla Public Licence (MPL), examined from a European and Dutch law perspective.

This paper is divided into three parts. Section 1 focuses on issues of copyright law. The open source software ideology, far from rejecting the rules of copyright law, relies on the

---

4 'Microsoft geeft ontwikkelaars meer inzicht', 8 February 2005, WebWereld, online: http://www.webwereld.nl/nieuws/20737.phtml (visited October 13th 2005).

5 S. Shankland, 'Adobe releases open-source interface software', 2 March 2005, CNET News.com.

6 Pearson, H.E., 'Open Source — The Death of Proprietary Systems?', 3 *Computer Law & Security Report* (2000), pp. 151-156, p. 152.

application of these rules to set their own 'open' terms of use of protected software. The key terms in open source software licences have been designed to take account of the fact that the traditional distinction between creators and users of works has essentially vanished thanks to the digital networked environment: users are creators and vice-versa. To accommodate the incremental development of creative works, the licences grant users the freedom to use, reproduce, modify the software, and the freedom to distribute or re-distribute the work. How do these freedoms fit in with rules on copyright? Section 2 examines the implications of the recognition of the patentability of software-implemented inventions for the development of open source software. To this end, we consider the patent protection as it is currently granted in Europe with respect to computer-implemented inventions. We also take a look at the reaction of some open source software developers in order to counter potential patent infringement claims from third parties. This includes the development of a patent strategy and the drafting of specific language such as the one appearing inside the GPL, and the Mozilla Public Licence. Section 3 takes a brief look at the enforcement of these licences. Finally, section 4 sets out a number of concluding remarks on the emergence of a new paradigm for the licensing of creative works.

1. Open Source and Copyright Law

Software developers across the European Union enjoy copyright protection on their programs since the adoption, in May 1991, of the Directive 91/250/EEC on the legal protection of computer programs.[7] As a result, all computer programs, whether in object code or in source code,[8] are subject to copyright protection in Europe, provided that they meet the habitual criterion of originality. Open source software does not differ in this respect from any other proprietary software. Open source software does, however, depart from proprietary software in the manner in which it is created and distributed to the public. The modes of creation and distribution of open source software have emerged in reaction to those of proprietary software, where the use of copyright law was seen as an impediment to the further development of software.[9] Far from rejecting the rules of copyright law, the open source movement relies on the application of these rules to set their own 'open' terms of use for protected software. The key terms in open source licences have been designed to take account of the fact that the traditional distinction between creators and users of software has essentially vanished within the open source community: users are creators and vice versa. In practice, the most widely used open source licences have been developed from an American law perspective, which shows important differences with European copyright law. In order for European users to be able to fully take part in the open source movement, it is paramount that each software developer knows precisely what his rights and obligations are under the law and the licence.

Since the adoption of the Computer programs directive, the right of reproduction is considered to include the permanent or temporary reproduction of a computer program by any means and in any form, in part or in whole. Given this very broad exclusive right of reproduction, some limitations had to be introduced to allow the lawful user to execute certain acts with respect to protected software without the former's prior authorisation. Nevertheless, most provisions in the national legislation concerning the use of computer programs are

---

7 Council Directive 91/250/EEC of 14 May 1991 on the legal protection of computer programs, *OJ* L 122 17.05.1991 p. 42 [hereafter the Computer programs directive].

8 Computer programs directive, recital 7, which reads as follows: "Whereas, for the purpose of this Directive, the term 'computer program' shall include programs in any form, including those which are incorporated into hardware; whereas this term also includes preparatory design work leading to the development of a computer program provided that the nature of the preparatory work is such that a computer program can result from it at a later stage".

9 Azzaria, G., 'Les logiciels libres à l'assaut du droit d'auteur', 16 *Les cahiers de propriété intellectuelle* (2004), pp. 405-428, p. 409.

merely default rules from which contracting partners may derogate. In practice, copyright owners usually grant users permission to accomplish certain acts with respect to their copyright protected work by means of a licence. The main difference between open source licences and proprietary software licences lies first and foremost in the freedoms that the former type of licence grants to users and, in certain cases, in the corollary obligation to make the source code available to fellow developers. As we will discuss below, the user of open source software enjoys an extended freedom to use, reproduce, modify and (re)distribute the software. In return, the licensee who undertakes to modify and redistribute new software based on an open source program must agree, at least under the GPL, to renounce receiving royalty payments for the use of the software. The following pages concentrate exclusively on the distinctions shown between the terms of the GPL, the BSD, and the Mozilla licences and the rules on copyright currently in force at the European level, as transposed into Dutch law.[10]

At the outset, it is important to note that, contrary to what is often asserted in the literature, the freedom of use afforded under most open source licences does not, as such, entail a waiver of right on the part of the "Rights Owner". In our opinion, the grant of a permission to execute certain acts with respect to a copyright protected work falls within the scope of the "Rights Owner's" exclusive right to authorise or prohibit the reproduction and communication to the public of his work and must be distinguished from a waiver of right. Admittedly, the line between a very broad licence to use a work and a waiver of right is in practice hard to draw. This may explain the confusion in terminology found in the literature. A waiver of right implies that the "Rights Owner" forsakes his right to exercise in the future one or all of his exclusive prerogatives, with respect to one or more individuals.[11] If the licensor waived his right of reproduction and communication to the public, there would logically be no consequence attached to the non-respect of the conditions by the licensee. However, by granting users the freedom to use, reproduce, modify and (re)distribute the open source software, the licensor does not relinquish his right to institute infringement proceedings should the licensee fail to honour the conditions set out in the licence. The failure by the licensee to meet the requirements of the GPL, or the Mozilla Public licence regarding the distribution of new versions of the software may result in the termination of the licence. This, in our opinion, supports the qualification of the open source licence as a broad permission, rather than a waiver of right.[12]

1.1 Freedom to use

Traditionally, copyright owners have never held absolute control over the use of their works. Everyone is in principle free to read, listen to, or view a work for his or her own learning or enjoyment. In theory, copyright never protected against acts of consumption or reception of information by individuals.[13] With the adoption of the Computer programs directive, this is no longer true, however, with respect to computer software.[14] In the case of software, the execution of even the most trivial operation constitutes a restricted act, since it involves making at least one temporary reproduction of the software in the RAM memory of the computer. Article 45i of the Dutch Copyright Act 1912 indeed specifies that "without prejudice to the provisions of article 13, the reproduction of a work as referred to in article 10,

---

10 For a more exhaustive account of the legal protection of proprietary software, we refer the reader to Meijboom, A. in Prins, J.E.J. et al., *Recht en Informatietechnologie – Handboek voor Rechtspraktijk en Beleid*, (Den Haag, SdU Uitgevers 1999), pp. 7I/1- 7I/36, pp. 7I/1- 7I/36; and Verkade, D.W.F., 'Intellectuele Eigendom', in H. Franken, H.W.K. Kaspersen and A.H. De Wild, *Recht en computer*, 5th ed., (Deventer, Kluwer 2004), pp. 227-289, pp. 242-258.

11 Spoor, J.H., D.W.F. Verkade and D.J.G. Visser, *Auteursrecht*, 3rd ed., (Deventer, Kluwer 2004), p. 552.

12 See: Clément Fontaine, M. *La licence publique générale gnu [logiciel libre]*, (Montpellier, Mémoire de D.E.A Droit des Créations Immatérielles, Université de Montpellier I, Faculté de Droit, 1999), online: http://www.freescape.eu.org/biblio/IMG/pdf/gpl.pdf, § 59.

13 Guibault, L.M.C.R., *Copyright limitations and contracts. An analysis of the contractual overridability of limitations on copyright* (The Hague/ London/Boston: Kluwer Law International 2002), p. 48.

14 Jaeger, T. & A. Metzger, *Open source software : Rechtliche Rahmenbedingungen der Freien Software*, (München, Verlag C.H. Beck, 2002), p. 21.

paragraph 1, sub 12°, shall include the loading, displaying, running, transmission and storage, in so far as these acts are necessary for the reproduction of the said work." According to recital 17 of the Computer programs directive however, the exclusive rights of the author to prevent the unauthorized reproduction of his work must be subject to a limited exception in the case of a computer program to allow the reproduction technically necessary for the use of that program by the lawful acquirer. According to article 5(1) of the Computer programs directive, in the absence of specific contractual provisions, the acts of reproduction referred to in article 4a) and b) do not require authorization by the right holder where they are necessary for the use of the computer program by the lawful acquirer in accordance with its intended purpose, including for error correction. This provision has been incorporated into article 45j of the Copyright Act 1912, which reads as follows:

> 'Unless otherwise agreed, the reproduction of a work as referred to in article 10, paragraph 1, sub 12° by the lawful acquirer of a copy of said work, where this is necessary for the use of the work for its intended purpose, shall not be deemed an infringement of copyright. Reproduction, as referred to in the first sentence, in connection with loading, displaying or correcting errors cannot be prohibited by contract.'

From the text of the directive and its implementing provision in the Dutch Copyright Act, it follows that the minimum rights of use are conferred only to the 'lawful acquirer' of a computer program. When is a person to be considered the 'lawful acquirer' of a computer program? More importantly for our purpose, can the person who downloads or otherwise obtains free of charge a copy of an open source program be seen as a 'lawful acquirer'? How must one interpret the 'lawful' character of the acquisition? Should the 'lawfulness' be assessed in relation to the authorisation to use the software granted under licence by the copyright holder, or in relation to the acquisition of the copy of the software, where the lawfulness is considered from a property law perspective.[15] In the first case, a user who acquires in good faith an infringing copy of the software would not be considered a 'lawful' acquirer of the program in the sense of the Copyright Act, while it could be true in the second case.

Van Schelven and Struik argue that, in view of the copyright dimension of the Computer programs directive, the 'lawfulness' of the acquisition should logically be evaluated from the perspective of the authorisation of the copyright holder rather than from a property law perspective. As a logical consequence of this, it is also generally accepted that a subsequent acquirer of the same copy of the software would be a 'lawful acquirer', even in the absence of a licence, according to the doctrine of exhaustion of rights.[16] In a preliminary ruling, the district court of Zutphen (The Netherlands) confirmed this interpretation of the expression 'lawful acquirer'.[17] The court ruled in this case that the simple fact that a copy of the program had been obtained legally, i.e., without having been stolen, it did not imply that the acquirer had the right to pose the acts of a 'lawful acquirer' in the sense of article 45j of the Copyright Act. On appeal, the Court of Arnhem overruled the decision, arguing among other things that since the software had been acquired in good faith the acquirer could be regarded as lawful within the meaning of article 45j of the Act.[18] This part of the court's ruling received severe criticism in the literature: first, the good faith character of the acquisition was irrelevant from

---

15 Van Schelven, P.C. and H. Struik, *Softwarerecht : bescherming en gebruik van programmatuur sedert de Richtlijn Softwarebescherming*, (Deventer, Kluwer 1995), p. 79-82.

16 Van Schelven and Struik 1995, *supra* note 15, p. 81; HR 25 January 1952, *NJ* 1952No. 95 (*Leesportefeuille*); and HR 20 November 1987, *NJ* 1988No. 82, with annotation from Wichers Hoeth (*Stemra/Free Record Shop*).

17 Arrondissementsrechtbank Zutphen, 29 April 1999 (*Deurwaarders Software Services*), *Computerrecht* 1999/4, § 9.19.

18 Gerechtshof Arnhem 11 december 2001 (*Blomsma/ Deurwaarders Software Services (DWSS) BV*), *Computerrecht* 2002/2, with annotation from E. Thole.

a copyright law point of view, since infringement done in good faith is still an act of infringement. Moreover, the appeals decision went against the majority opinion which considers the 'lawful acquirer' solely to be the one who is authorised to use the software in accordance with a purchase or licence contract from the copyright owner or his assignee.

The European Commission would seem to agree with the majority of opinion in the Netherlands, which considers the 'lawful acquirer' to be the one who is authorised to use the software in accordance with a purchase or licence contract from the copyright owner or his assignee and not to be the one who legally obtained a copy of the program. In its report on the implementation and effects of Directive 91/250/EEC on the legal protection of computer programs, the European Commission observes that divergences of views subsist however as to the meaning of 'lawful acquirer'. Several Member States have transposed this notion by using the term 'lawful user' i.e., a person having a right to use the program. The Commission shares the view of some commentators that 'lawful acquirer' does in fact mean a purchaser, licensee, renter or a person authorised to use the program on behalf of one of the above. This argument also draws from Articles 6 and 8 of the Database Directive (Directive 96/9/EC)[19], which use the term 'lawful user', and which were modelled along the lines of Article 5 (1) of the Computer Programs Directive. In the view of the Commission, what was intended by Article 5 (1) and recital 18 was that it should not be possible to prevent by contract a "lawful acquirer" of a program doing any of the restricted acts that were required for the use of the program in accordance with its intended purpose or for correcting errors. It is, however, possible for a contract to include specific provisions that "control" the restricted acts, which may be carried out by the user of the computer program.[20]

With this definition of a 'lawful acquirer' in mind, one could reasonably argue that anyone having a licence to use an open source computer program is a 'lawful acquirer' of that program within the meaning of the Copyright Act, provided that the licence accompanying the product was validly entered into. Whether the user of the open source software obtained the copy free of charge or not should make no difference, since the majority opinion considers that the lawfulness of the acquisition should not be assessed from a property law perspective. In other words, whether the transaction would qualify as a donation rather than a purchase is irrelevant for the determination of when a user is a 'lawful acquirer' of open source software pursuant to article 45j of the Act.

According to some commentators, a literal interpretation of article 45j of the Act would suggest that a person who downloads an electronic version of the software instead of acquiring a tangible copy of the same software could not be regarded as a lawful acquirer of 'a copy' of the software. As a result, the acquirer of a computer program downloaded from the Internet would not be entitled to benefit from the minimum rights of use.[21] In our opinion, it cannot have been the intent of the legislator to limit the application of the provision according to the medium upon which the software is distributed to the public. Arguably, at the time the directive was adopted in 1991, neither the European legislator nor the national legislators of the Member States may have been aware of the possibility to distribute software on-line, as an economically viable mode of exploitation.[22] Today, on-line distribution has

---

19 Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the Legal Protection of Databases, O.J. L 077, 27 March 1996, pp. 20-28.

20 Report from the Commission to the Council, the European Parliament and the Economic and Social Committee on the implementation and effects of Directive 91/250/EEC on the legal protection of computer programs, COM/2000/199 final, Brussels, April 10th 2000, online: http://europa.eu.int/eur-lex/lex/LexUriServ/site/en/com/2000/com2000_0199en01.pdf.

21 Koelman, K.J., "Brothers in arms: open source en auteursrecht", 5 *Computerrecht* (2004), pp. 230-233, p. 230; Thole, E. and W. Seinen, 'Open Source Software licenties: een civielrechtelijke analyse', 5 *Computerrecht* (2004), pp. 221-225, p. 221; and M.W. Scheltema and E. Tjon Tjin Tai, "Overeenkomsten sluiten door openen en klikken?", 4 *Computerrecht* (2003) pp. 244-248, p. 248.

22 See: Proposal for a European Parliament and Council Directive on the harmonisation of certain aspects of copyright and related rights in the Information Society, Explanatory Memorandum, December 1997, §

become one of the most important modes of exploitation of both proprietary and open source software, including for popular programs like Microsoft Windows and Linux. Such a restrictive and technology dependent interpretation would, in our opinion, be inconsistent with the more common interpretation of the provision according to which 'lawful acquirer' means the person who is authorised to use the program. From a practical point of view, this interpretation would also frustrate the reasonable expectations of use of all licensees who acquire software on-line, consequence, which can hardly be justified under the law.

Article 45j of the Dutch Copyright Act also implies that while "Rights Owner's" may contractually regulate the running, transmitting or storing of a computer program, they may not prohibit lawful acquirers from performing such acts as the loading, displaying or correcting of errors. The last sentence makes it clear that, in view of the unprecedented expansion of the copyright protection, the Dutch legislator wanted to guarantee a minimum right of the lawful acquirer of a copy of a computer program to perform those acts that are necessary for the normal use of the computer program.[23] Apart from the limited acts of loading, displaying, or correcting errors, a lawful acquirer may, however, only execute those acts that are necessary for the use of the work for its intended purpose. When can an act be deemed necessary for the use of the program for its intended purpose? Verkade notes on this subject that the circular formulation of articles 45i and 45j is the result of a political compromise and that it certainly cannot have been the intention of the legislator to include any and all technically possible acts of reproduction within the scope of protection of the "Rights Owner".[24] For Meijboom, the intended purpose of the software is a question of fact that can be assessed in relation to the software's nature, functionality, or capacity. The decisive factor in establishing what the intended use of a particular program is consists in looking at the common intention of the parties at the time they concluded the licensing agreement.[25] When the interpretation of the licence contract offers no concrete solution, the intended purpose of the software can be estimated in function of the use that the average purchaser could reasonably have expected to make of the software.[26]

It is worth pointing out in this context that the Directive on copyright and neighbouring rights in the information society[27] introduced a mandatory exception for temporary acts of reproduction. Article 13a of the Copyright Act, which transposed this last provision into Dutch law, provides that:

> "The reproduction of a literary, scientific or artistic work does not include temporary acts of reproduction which are transient or incidental and an integral and essential part of a technological process and whose sole purpose is to enable: (a) a transmission in a network between third parties by an intermediary, or (b) a lawful use, of a work or other subject-matter to be made, and which have no independent economic significance."

II.A.5, where the Commission writes: 'The Computer Programs Directive, in its Article 4, however, only protects "any form of distribution to the public" of computer programs, not expressly addressing its on-line transmission over the networks. Indeed, at the time of adoption of the Directive the usual form of distribution took place on the basis of floppy discs and not on-line'. And Meijboom, 'The EC Directive on Software Copyright Protection', in Jongen, H.D.J. and A.P. Meijboom (eds.), *Copyright Software Protection in the EC*, (Deventer, Kluwer Law and Taxation Publishers 1993), p. 11 where the author writes: "Usually, software is transferred, in whole or part, from the medium on which it is supplied (diskette, tape, hard disk) to the computer's memory in order to execute the program".

23 Guibault 2002, *supra* note 13, p. 220.

24 Verkade 2004, *supra* note 10, p. 253.

25 Meijboom 1999, *supra* note 10, p. 71 – 25.

26 Verkade 2004, *supra* note 10, p. 254.

27 Directive 2001/29/EC of the European Parliament and of the Council of 22 May 2001 on the harmonisation of certain aspects of copyright and related rights in the information society, L 167, 22/06/2001, p 10-19 [hereafter InfoSoc directive].

According to recital 50 of the InfoSoc directive, however, 'such a harmonised legal protection does not affect the specific provisions on protection provided for by Directive 91/250/EEC. (...) Articles 5 and 6 of that Directive exclusively determine exceptions to the exclusive rights applicable to computer programs.' In other words, the temporary reproduction of any other type of copyright protected work than computer software is excluded from the scope of the "Rights Owner's" exclusive right, provided that the conditions of application of article 13a of the Act are met. As some commentators have argued, the Dutch legislator would have been wise to review article 45i in the light of new article 13a of the Act in order to avoid any possible ambiguity.[28]

Under Dutch copyright law, pure consumptive uses of computer programs such as loading, displaying and correcting errors on the software cannot be prohibited by contract, even if they technically fall under the scope of the owner's exclusive right. However, the licensor is allowed to contractually regulate the running, transmitting or storing of a computer program. In this sense, open source licences grant users a much greater freedom of use than article 45j of the Act. This is particularly evident from article 0 of the GPL, which specifies that "Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does." The BSD licence states that redistribution and use in source and binary forms, with or without modification, are permitted provided that certain conditions regarding the redistribution of software are met. Similarly, article 2.1 of the Mozilla Public Licence grants the user the following rights: "a world-wide, royalty-free, non-exclusive license, subject to third party intellectual property claims: (a) under intellectual property rights (other than patent or trademark) Licensable by Initial Developer to use, reproduce, modify, display, perform, sublicense and distribute the Original Code (or portions thereof) with or without Modifications, and/or as part of a Larger Work".

Contrary to what article 5 of the GPL states, we believe that consumers should obtain a valid licence for the use of the software. For, without a licence, consumers are in principle restricted to the acts mentioned in the Copyright Act. For example, article 45j of the Act permits a lawful acquirer to perform only those acts that are necessary for the use of the work for its intended purpose, apart from the limited acts of loading, displaying, or correcting errors. Moreover, even the making of private copies of software, let alone their distribution among friends and family, is strictly prohibited under the law. The freedom of use and reproduction granted under a typical open source licence is generally much broader than what is allowed under copyright law, making the need to obtain a valid licence if not necessary at least recommendable.

1.2 Freedom to reproduce

As soon as a user wants to do more with his software, than merely loading and displaying it on his computer, he must as a matter of course make a reproduction of the program. This is true not only for running, transmitting, or storing a computer program, but also for ensuring its maintenance[29] and for translating or adapting it. Unless these acts are covered by a limitation on copyright, the user is obligated to obtain permission from the rights holder prior to making any kind of reproduction. The Dutch Copyright Act, on the model of the Computer

---

[28] Kleve, P., *Juridische iconen in het informatietijdperk*, (Deventer, Kluwer 2004), p. 210.

[29] Rb's–Gravenhage 23 April 2003, *Computerrecht*, 2004/5 (*Faco Informatisering BV/ Haley Software BV*), § 8.

programs directive, grants the lawful user[30] only limited rights to make unauthorised reproductions of protected computer programs. Article 45k of the Act allows the lawful user of a program to make a copy of that program to serve as a back-up copy, where this is necessary for the use of the work for its intended purpose. The making of private copies of a program is strictly prohibited under article 45n of the Copyright Act.[31] Article 45l states that a person who is entitled to perform the acts referred to in article 45i shall also be entitled, while performing them, to observe, study or test the functioning of the work concerned in order to determine the ideas and principles underlying it. Article 45m permits the making of a copy of a program and the translation of the form of its code, provided that these acts are indispensable for obtaining information necessary to achieve the interoperability of an independently created computer program with other programs, and provided that a number of conditions are met. In the Explanatory Memorandum to the Implementation Act, the Dutch government did indicate that the limitations on the exclusive right, such as those set out in Articles 45k, 45l, and 45m of the Act, were imperative. However, according to the government, there was no need to specify this in the Act.[32] Although it would certainly have been clearer to spell it out in the Act, the Dutch courts cannot ignore the mandatory character of these provisions, since they too must interpret these provisions in compliance with the directive.[33] A great deal has been written concerning the scope of these limitations with respect to proprietary software.[34] Suffice to say, here, that the general limitations on the owner's exclusive rights, such as the right to quote and the right to use work for educational purposes are also applicable with respect to the reproduction of a computer program.[35]

In light of these provisions, it is clear that once again the GPL, the BSD, and the Mozilla Public licence all offer the licensee much greater freedom to reproduce the computer program, without restriction as to the number of copies realised or to the purpose for making these copies. From the perspective of the licensor(s), these licences are valid, as "Rights Owner's" are entitled to licence their rights as they see fit. Nothing in the Dutch Copyright Act prevents "Rights Owner's" to license their rights broadly to a third party, whether on an exclusive or non-exclusive basis, for a fee or for free. The only restrictions on the freedom of contract of the "Rights Owner" would be set by the imperative character of the limitations relating to the making of a back-up copy, of a reproduction for purposes of observing, studying and testing the software, as well as to the decompilation of the program for purposes of interoperability. These restrictions are irrelevant in the context of open source software, since all types of open source licences grant the user much broader rights of use than the law normally does.

1.3 Freedom to modify

Generally speaking, the right to modify, adapt, or transform a protected work falls under the exclusive right of reproduction of the owner. This principle is derived from article 13 of the Dutch Copyright Act, which provides that "the reproduction of a literary, scientific or artistic work includes the translation, arrangement of music, cinematographic adaptation or dramatization and generally any partial or total adaptation or imitation in a modified form,

---

30 According to Meijboom [1999, *supra* note 10, p. 7I – 28], the expression 'lawful user' should be understood in the same terms as the expression 'lawful acquirer' since a person may only be a lawful user if he is a lawful acquirer in the sense of articles 45*i* and 45*j* of the Act.

31 Report from the Commission to the Council, the European Parliament and the Economic and Social Committee on the implementation and effects of Directive 91/250/EEC on the legal protection of computer programs, COM/2000/0199 final.

32 D.W.F. Verkade, "Computerprogramma's in de Auteurswet 1912: het vierde regime...", 3 *Computerrecht* (2003), pp. 86-97, p. 95.

33 Guibault 2002, *supra* note 13, p. 218.

34 Verkade 2004, *supra* note 10, p. 252 et seq.; Meijboom 1999, *supra* note 10, p. 7I – 18 et seq.

35 Groenenboom, M.M., 'Software licenties: van closed source tot open source', 1 *Computerrecht* (2002), pp. 21-29, p. 22.

which cannot be regarded as a new, original work". The rights holder's exclusive right of reproduction entails more than just the right to authorise or prohibit the making of exact or substantially similar copies. It also extends to the making of arrangements, adaptations, and modifications to an existing work, otherwise called 'derivative works'. Any arrangement, adaptation, or modification of an existing work is subject to the prior authorisation of the rights holder.[36] With respect to computer programs, recital 20 of the Computer programs directive states: "the unauthorized reproduction, translation, adaptation or transformation of the form of the code in which a copy of a computer program has been made available constitutes an infringement of the exclusive rights of the author". Besides the generally applicable limitations on copyright, such as the right to quote and to make reproductions for purposes of research and private study, no specific limitation tempers article 45i of the Copyright Act, with respect to the translation, adaptation, or transformation of software. In other words, a computer program may not be translated, adapted, or transformed without the rights holder's prior authorisation. Moreover, proprietary licensing contracts are usually adamant in requiring that the licensee refrain from bringing any modification to the software without prior authorisation from the rights holder.

Perhaps more than the freedom to use or to reproduce a computer program, the freedom to modify the software constitutes the cornerstone of the open source movement. As one can read on the home page of the Open Source Initiative (OSI):

> "The basic idea behind open source is very simple: When programmers can read, redistribute, and modify the source code for a piece of software, the software evolves. People improve it, people adapt it, people fix bugs. And this can happen at a speed that, if one is used to the slow pace of conventional software development, seems astonishing. We in the open source community have learned that this rapid evolutionary process produces better software than the traditional closed model, in which only a very few programmers can see the source and everybody else must blindly use an opaque block of bits."[37]

As a rule, users of open source software have, pursuant to the GPL, the BSD, or the MPL, the right to modify the software and to prepare derivative works based upon the original work. Indeed a particular computer program may be qualified as 'open source software' only if the licence allows modifications and derivative works, and allows them to be distributed under the same terms as the license of the original software. To facilitate the modification and the evolution of computer programs, most open source licences require that the source code be distributed along with the object code of the program, or that it at least be made available to the public. Only by having access to the source code of an existing computer program, are software developers in a position to build upon that existing program in order to improve it or to develop compatible software.[38] Proprietary software manufacturers are usually very protective of their source code, for it may embody competitive trade secrets, and only gives users access to it in rare circumstances and then, only under controlled conditions.[39] In the same vain, proprietary software suppliers are generally highly reluctant to provide interested parties with interface information. Without the proper interface information or the possibility to decompile a program, computer programmers are absolutely unable to develop any kind of software that is interoperable with existing software. At the time of the adoption of the Computer programs directive, the question of whether the decompilation of a program should be allowed led to heated debates. As a result, the directive contains a mandatory limitation allowing under strict conditions lawful users to decompile a program for purposes of

---

36 Van Lingen, N., *Auteursrecht in hoofdlijnen*, 5th ed., (Groningen, Martinus Nijhoff 2002), p. 77.

37 See <http://www.opensource.org/>, site visited on 23 November 2004.

38 Guadamuz Gonzalez, A., 'Viral Contracts Or Unenforceable Documents? Contractual Validity Of Copyleft Licences', *E.I.P.R.* (2004), pp. 331-339, p. 332.

39 Hoeren, T., 'Die Pflicht zur Überlassung des Quellcodes', 10 *Computerrecht* (2004), pp. 721-724, p. 721.

interoperability, which the Dutch legislator has transposed in article 45m of the Copyright Act.[40]

According to article 3 of the GPL, for an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable code. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable code runs, unless that component itself accompanies the executable code. If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source code along with the object code.

Decompilation of a program becomes no more than a useless intellectual challenge in the context of open source software. The first principle laid down in the Open Source Definition (OSD) holds that 'the program must include source code, and must allow distribution in source code as well as in a compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost–preferably, downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a pre-processor or translator are not allowed.' The OSD's second principle concerns derivative works, whereby 'the license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.' The rationale behind this principle is that the mere ability to read the source code is not enough to support independent peer review and rapid evolutionary selection. For rapid evolution to happen, people need to be able to experiment with and redistribute modifications. What constitutes a modification of an open source computer program must be evaluated, in the same way as for any other type of work, according to the criterion of originality.

While neither the GPL nor the BSD licence gives any definition of what must be understood by 'modification', the Mozilla license defines 'modifications' as follows: 'any addition to or deletion from the substance or structure of either the Original Code or any previous Modifications. When Covered Code is released as a series of files, a Modification is: A. Any addition to or deletion from the contents of a file containing Original Code or previous Modifications. B. Any new file that contains any part of the Original Code or previous Modifications.' Arguably, even if they do not expressly define it, the GPL and the BSD licences should make reference to a similar notion of 'modification', presumably derived from the American case law on the notion of 'derivative works'.[41] The BSD and the Mozilla licences grant the user comparable freedom to make modifications to existing open source software. However, article 10 of the GPL warns the user that if he wishes to incorporate parts of the licensed program into other free programs whose distribution conditions are different, he must write to the author to ask for permission. In the case of software licensed by the Free Software Foundation, exceptions can be made, each case being assessed following the double objective of preserving the free status of all derivative works based on free software and of promoting the sharing and reuse of software generally.

---

40 Van Lingen 2002, *supra* note 36, p. 62; De Cock Buning, M., 'Auteursrecht en reverse engineering', 5 *IER* (1993), p. 129-137, p. 129; and Meijboom in Jongen and Meijboom 1993, *supra* note 22, p. 14.

41 See Title 17 U.S.C. § 101, definition of 'derivative work': A "derivative work" is a work based upon one or more pre-existing works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which a work may be recast, transformed, or adapted. A work consisting of editorial revisions, annotations, elaborations, or other modifications, which, as a whole, represent an original work of authorship, is a "derivative work".

As we shall see in the next section, the freedom to modify open source software under all types of licences is further accompanied by strict obligations as soon as the user wishes to distribute software based on software originally distributed under an open source licence.

1.4 Freedom to (re)distribute

The freedom to redistribute copies of the software or to distribute a modified version of the software is, just as the freedom to make modifications, one of the key features of any open source licence. A particular computer program will fall under the Open Source Definition only if it complies with the first principle laid down by the Open Source Initiative (OSI), according to which the 'license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources.' As a rule, open source licences therefore afford users much greater freedom than article 12 of the Copyright Act normally would with regard to the right to distribute a copy of a copyright protected work, since the exclusive right of the rights holder includes any form of distribution to the public, including the rental, of the original computer program or copies thereof.[42] The only exception to this rule is the one provided for in article 15b of the Act, which concerns the further communication to the public or the reproduction of a literary, scientific or artistic work communicated to the public by or on behalf of the public authorities. According to this provision, unless the copyright has been explicitly reserved, either in a general manner by law, decree, or ordinance, or in a specific case by a notice on the work itself or at the point of communication to the public, a work that is communicated by or on behalf of the public authorities may be freely distributed. This exception would be applicable for example in the case of software distributed by the government.

Under the three types of open source licences examined here, the exercise of this freedom is accompanied by a number of strict conditions of application. The BSD licence states for instance that redistributions of source code must retain the copyright notice, the list of conditions and a disclaimer. Redistributions in binary form must reproduce the copyright notice, the list of conditions and a disclaimer in the documentation and/or other materials provided with the distribution.

As we have seen in the previous subsection, one of the main conditions under the GPL is that the source code be distributed along with the program or that it be made available to any third party who requests it. Article 2 of the GPL regulates essentially the same elements as the BSD licence, i.e., the placement of a copyright notice, of a list of conditions and a disclaimer of warranty and liability, but in much greater detail. As the GPL explains, "it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program." The GPL licence goes further than the BSD licence in that it requires the user to cause any work that he or she distributes or publishes, that in whole or in part contains or is derived from the program, to be licensed as a whole at no charge to all third parties under the terms of the GPL License.

According to article 2 of the Mozilla licence, the initial developer grants the user a licence to among other things distribute the original code (or portions thereof) with or without modifications, and/or as part of a larger work. Article 3 of the Mozilla licence sets out "distribution obligations", which are comparable in length and complexity to those of the GPL. The provision requires among other things that the user distribute any copy of the software or any work derived from the original code only under the terms of this licence and that a copy of the licence be included in the distribution. The user must also make any

---

42 Van Lingen 2002, *supra* note 36, p. 87.

modification that he creates available in source code. He must also document the changes made to the original software and duplicate the prescribed copyright notice.

In this sense, the GPL and the Mozilla licences follow the OSI's foremost precept, according to which "the rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties". This clause, known as the 'copyleft' clause, is intended to forbid closing up software by indirect means such as requiring a non-disclosure agreement. To this end, the distribution obligations are placed not only on the initial licensee, but also on any subsequent licensee. In practice, the 'copyleft' clause varies in scope from one type of open source license to another and, as the BSD licence demonstrates, not every type of open source licence contains such a clause.[43]

Under the GPL and the Mozilla licences, the copyleft clause is applicable to the distribution of the original code with or without modification. In the case of the distribution of modified code, the question can arise, however, whether the product involved actually does constitute a 'derivative' work based on the original work, or if it rather constitutes an entirely new work, in the sense of article 13 last sentence, of the Dutch Copyright Act 1912.[44] For, if the new software qualifies as a new work under the copyright act, the developer is in principle not bound by the copyleft term of the licence. As McGowan rightfully observes, there are problems with the proposition that a person creates a work derivative of a program just by writing another program that interacts with it.[45] The last part of article 2 of the GPL does specify the following:

> "These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.
> Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.
> In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License".

In the open source context, the discussion has focused essentially on the question of whether static and dynamic linking to a computer program distributed under the GPL entail, as a consequence, that the linking program must also be distributed under the GPL. The Free Software Foundation (FSF) has adopted the position that both static and dynamic linking to a computer programme may result in a "derivative" work.[46] Whether other combinations of code modules must also be regarded as an arrangement depends, according to the FSF, on the communication mechanism and the semantics of the communication between the modules.

---

43 O'Sullivan, M., 'The Pluralistic, Evolutionary, Quasi-legal Role of the GNU General Public Licence in Free/Libre/Open Source Software ("FLOSS")', *EIPR* 2004, pp. 340-348, p. 343.

44 Verkade 2004, *supra* note 10, p. 251.

45 See for example: McGowan, D., 'Legal Aspects of Free and Open Source Software', in J. Feller, B. Fitzgerald, S. Hissam and K. Lakhani (ed.), *Perspectives on Free and Open Source Software*, (MIT Press, Boston, MA, 2005), pp. 361-392, online: http://www.law.umn.edu/FacultyProfiles/articles/McGowanD-OpenSource.rtf. (p. 23).

46 Free Software Foundation, <http://www.gnu.org/licenses/gpl-faq.html#MereAggregation>. Incidentally, this rather broad interpretation of the term "derivative work" may have as a consequence that emulators like Wine, which use the libraries of Windows to function, would infringe Microsoft's copyright.

The fact that modules are combined into the same file suggests that we are in presence of an adaptation or an arrangement. If the modules are developed in order to be executed together in a combined address location, we can speak of an adaptation or an arrangement. However, collective communication mechanisms, which are normally used by two separate programmes, will not easily be seen as an adaptation. On the other hand, whenever the semantics of the communication are sufficiently "proximate", one can conclude that an adaptation has been realised. In all cases, the question of whether the new code forms a derivative work or an entirely new work is a matter of fact that should be decided on a case-by-case basis.

The copyleft clause may raise problems from the point of view of copyright law. A potential problem concerns the application of the doctrine of exhaustion of rights. According to article 4c) of the Computer programs directive, "the first sale in the European Community of a copy of a program by the right holder or with his consent exhausts the distribution right within the Community of that copy, with the exception of the right to control further rental of the program or a copy thereof". As a consequence of this rule, the author of a work loses control over the further dissemination of the copy after it has been made public by him or with his consent. Therefore, selling, lending, leasing and hiring of a copy of a computer program by the lawful acquirer cannot be prohibited.[47] The application of this doctrine has been reaffirmed in slightly different words in article 4(2) of the InfoSoc Directive. This provision states that "the distribution right shall not be exhausted within the Community in respect of the original or copies of the work, except where the first sale or other transfer of ownership in the Community of that object is made by the right holder or with his consent". Does the distribution of computer programs under the terms of an open source licence constitute a 'sale' in the sense of the Computer Programs Directive, which would have as a consequence the effect of exhausting the distribution right of the rights holder? A corollary question to this is whether the distribution of software free of charge entails a 'transfer of ownership', which would lead to an exhaustion of right under Community law? Another corollary question is whether, for the purposes of the exhaustion doctrine, there is a difference between the off-line or on-line distribution of a 'copy' of a computer program.

Software manufacturers often maintain that the distribution right is not exhausted through the grant of a licence of use of the software, because the licensing of rights does not constitute "the first sale in the European Community of a copy of a program by the right holder or with his consent". This argument could be inferred from the wording of article 4c) of the Computer programs directive, which would seem to limit the application of the exhaustion doctrine to the 'sale' of a computer program, whereas any other form of distribution would not give rise to the application of the doctrine.[48] This theory has been, in our opinion, rightfully contested.[49] Along with Neppelenbroek, we believe that the exhaustion doctrine does not so much focus on the concept of 'sale', but rather on that of 'transfer of ownership'. As Grosheide explains with respect to the general principle of exhaustion of rights: "it is not limited to the first sale of the copy but encompasses other forms of distribution such as donation and first rental. Exhaustion assumes assignment of title with regard to the copy (i.e., the content carrier)."[50] This interpretation of the exhaustion doctrine would, in our opinion, be more consistent with the interpretation of the doctrine as it is set out in other European

---

47 Jongen, in Jongen and Meijboom 1993, *supra* note 22, p. 174. See HR 25 January 1952, *NJ* 1952No. 95 (*Leesportefeuille*); and HR 20 November 1987, *NJ* 1988No. 82, with annotation from Wichers Hoeth (*Stemra/Free Record Shop*).

48 Van Schelven and Struik 1995, *supra* note 15, p. 70-71; and Grosheide, F.W., 'Mass-market exploitation of digital information by the use of shrink-wrap and click-wrap licenses', in F.W. Grosheide & K. Boele-Woelki (ed.), *Opstellen over Internationale Transacties en Intellectuele Eigendom*, (Lelystad, Koninklijke Vermande 1998), Molengrafica Series, pp. 263-319, p. 308.

49 Neppenlenbroek, E.D.C., 'Software en de uitputtingsregel', 6 *AMI* (2001), pp. 125-132, p. 126.

50 Grosheide 1998, *supra* note 48, p. 307.

Directives in the field of copyright. It follows from this that in any case, the mere labelling of a transaction as a licence is insufficient as such to circumvent the exhaustion doctrine.

The question of whether the grant of a licence can amount to a sale or to another form of distribution giving rise to the application of the exhaustion doctrine is a matter of fact that should be decided on a case-by-case basis. In the Netherlands, opinions are divided on whether the distribution to the public of a computer program on a tangible medium (i.e., floppy disc or CD-ROM) for an unlimited term and an outright fee is more akin to a sale than a licence, understood in the strict meaning of the word. Such a transaction would entail, in our opinion, a transfer of ownership of the physical embodiment of the work, which would lead to the application of the exhaustion doctrine. In this sense, the court of appeal of The Hague once ruled that the view, according to which the further distribution of the software can be blocked through a clause prohibiting further transfers, would unduly restrict the working of the exhaustion doctrine.[51] In Germany, it is generally accepted that the distribution right is exhausted as soon as a computer program is put into circulation following the terms of a licence and against the payment of a one-time fee, a position that was confirmed by the Federal Supreme Court in the OEM-Version case.[52] The situation would be different if the licence to use the software was limited in time and if the licensee was obligated to periodically pay a fee during the entire duration of the licence. In this case, there would be no transfer of ownership of the physical embodiment of the work, and the distribution right of the rights holder would not be exhausted. On the other hand, the licensing of a copy of the software for an indefinite term, but free of charge, would probably qualify as a donation, thereby implying a transfer of ownership of the physical embodiment of the work.[53] As a result, the distribution right of the rights holder would be exhausted as soon as a tangible copy of the work is put into circulation, even if this occurs free of charge. As Spindler observes, the application of the exhaustion doctrine does not depend on whether the copy of the work is distributed for a price or free of charge. The important factor is that, through the granting of a licence, the distributor operates a definitive transfer of ownership of the software in favour of the licensee.[54]

The above remarks concern the distribution of physical copies of computer programs, i.e., on floppy discs, CD-ROM's, and the like. To the question formulated above, of whether, for the purposes of the exhaustion doctrine, a difference must be made between the off-line or on-line distribution of a 'copy' of a computer program, the answer is yes. While the non-application of the exhaustion doctrine to the electronic delivery of computer programs could already be inferred from the wording of article 4c of the Computer programs directive, which refers only to the "first sale in the European Community of a 'copy' of a program", the question would seem to have been resolved at the European level. According to Recital 29 of the InfoSoc directive:

> "The question of exhaustion does not arise in the case of services and on-line services in particular. This also applies with regard to a material copy of a work or other subject matter made by a user of such a service with the consent of the right holder. Therefore, the same applies to rental and lending of the original and copies of works or other subject matter, which are services by nature. Unlike CD-ROM or CD-I, where the intellectual property is incorporated in a material medium, namely an item

---

51 E.D.C. Neppelenbroek, annotation by Hof 's-Gravenhage 20 November 2003 (*Ist Flight Training*), *Computerrecht* 2004/3.

52 BGH, 6 July 2000, I *ZR* 244/97 (*OEM-Version*), *Computer und Recht* 2000, p. 654; see also: Jaeger and Metzger 2002, *supra* note 14, p. 22; and Spindler, G., *Rechtsfragen der Open Source Software*, (Göttingen, Verband Software Industrie 2003), available at: http://www.vsi.de/inhalte/aktuell/studie_final_safe.pdf., p. 48.

53 See Dutch Civil Code, art. 7:175.

54 Spindler 2003, *supra* note 52, p. 51, footnote 302.

> of goods, every on-line service is in fact an act which should be subject to authorisation where the copyright or related right so provides."

The notion that the electronic distribution of works does not give rise to the exhaustion doctrine because it falls under the scope of the right of making a work available to the public, rather than under the right of distribution, is now part of the *acquis communautaire*.[55] For more certainty, the European Commission clearly stated, in its report on the implementation of the Computer programs directive, that community exhaustion only applies to the sale of copies, i.e., goods, whereas supply through on-line services does not entail exhaustion.[56] Although this distinction may be unfortunate in the eyes of some commentators,[57] we will not dwell on the issue any further. Nevertheless, it could be argued that the exhaustion doctrine could apply to the tangible copy made from a digital version of a computer program downloaded from the Internet. It would indeed not be unreasonable to think that, if the lawful acquirer of an electronic version of computer program burned the software on a CD, he would be able to transfer that specific CD to a third party without infringing the owner's copyright, provided that the initial copy of the programme is deleted from his computer.

In light of all this, let us now consider how the exhaustion doctrine applies in the case of an open source licence. We will recall that, under the GPL and the Mozilla licences, the 'distribution obligations' are applicable to the distribution of the original code with or without modification. One must realise at this point that the doctrine of exhaustion applies only to the distribution right, not to the right of reproduction or to the right to distribute derivative works.[58] The distribution right is therefore subject to exhaustion only in the case where the original software is distributed on a tangible medium, i.e., on floppy discs or CD-ROM's, where the licence terms can be interpreted as operating a transfer of ownership in the software and where the licensee further distributes exact and unmodified copies of that software. In such circumstances, the 'distribution' obligations of the GPL, the BSD, or the Mozilla Licence would not be binding upon the licensee. On the other hand, these obligations would be binding upon the licensee whenever the open source software is delivered on-line, or when the licensee creates and distributes a derivative work based on an open source computer program. In practice, the exhaustion doctrine would come into play only in limited circumstances, since the large majority of open source software is distributed over the Internet and since a software developer has little interest in distributing exact copies of a program that is otherwise freely available elsewhere. A computer programmer will be much more inclined to put improved versions of the software into circulation. In that case, he must comply with the requirements of the licence, namely to distribute the source code along with the object code of the program, or at least to make it available to the public, to put the proper copyright notices and, in the case of the GPL, to distribute the modified software under the same licence terms.

1.5 Royalty free distribution

In addition to the threefold requirement mentioned above, open source licences generally demand that the software developer, who wishes to distribute a modified version of the open source software, agree not to require a royalty or other fee for the sale of open source software. By imposing this requirement, the Open Source Initiative hopes that the temptation to throw away many long-term gains in order to make a few short-term sales dollars will

---

55 Walter, M.M. (ed.), *Europäisches Urheberrecht – Kommentar*, (Wien, New York, Springer Verlag 2001), p. 1053; and Neppelenbroek 2001, *supra* note 49, p. 127.

56 Report from the Commission to the Council, the European Parliament and the Economic and Social Committee on the implementation and effects of Directive 91/250/EEC on the legal protection of computer programs, COM/2000/0199 final.

57 Bolcher in Walter 2001, *supra* note 55, p. 171 and ff.; Tjon Tjin Tai, E., 'Exhaustion and Online Delivery of Digital Works', *EIPR* (2003), pp. 207-210, p. 207.

58 Walter 2001, *supra* note 55, p. 1043.

disappear.[59] Otherwise, the OSI fears that co-operators would find themselves under a lot of pressure to defect from the open source movement in favour of more lucrative activities. In practice, this condition means that the distribution of any software developed on the basis of an open source programme must not be subject to the payment of a royalty fee.[60] Not all licences contain this requirement. Whenever they do not, like the BSD and the Mozilla licences, they are not regarded as falling under article 1 of the Open Source Definition which states that 'the license shall not require a royalty or other fee for such sale'. However, the prohibition to charge royalties for the use of the software does not exclude the possibility of charging a fee for other aspects of the distribution. Article 1 of the GPL stipulates, for example, that "you may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee."

The obligation to license open source software free of charge raises little concern from a copyright law perspective. Pursuant to the Dutch Copyright Act, an author may transfer his exploitation rights in whole or in part; or grant them under licence, exclusive or non-exclusive, for a fee or free of charge. Nothing in the Copyright Act precludes rights holders from agreeing by contract not to ask for royalties for their copyright protected work.[61] From Microsoft to small software developers, the free distribution of software is in fact very common.[62] Most of all, the GPL does not prevent "Rights Owner's" to make money, for example by charging for support and for guarantee. As an illustration of this, the Linux operating system is distributed, free of charge, over the Internet under the GPL terms. However, for any user who feels unable to download the free version of the software from the Internet and install it, potential users have the possibility of purchasing from several vendors a boxed set with the Linux system. In this case, the cost of Linux system increases, but on the other hand, the user will be provided with full documentation, step-by-step installation instructions and in many cases free technical support for up to 90 days by phone or e-mail.[63]

While the obligation to distribute software on a royalty free basis would probably pose no difficulty from a copyright law point of view, such an agreement may, in certain circumstances, raise competition law concerns. A first concern relates to the obligation of open source licensees to distribute the software free of charge. It has been suggested that this practice could, in certain circumstances, be regarded as an unlawful imposition of a vertical restraint in the form of resale price maintenance, contrary to article 6 of the Dutch Competition Act or article 81(1) of the European Union Treaty.[64] Both provisions prohibit all agreements between undertakings, which have as their object or effect the prevention, restriction, or distortion of competition within the common market, including agreements that directly or indirectly fix purchase or selling prices. However, a restrictive agreement may still be allowed, under article 6(3) of the Dutch Competition Act and article 81(3) of the European Treaty, if it contributes to improving the production or distribution of goods or to promoting technical or economic progress, while allowing consumers a fair share of the resulting benefit, and which does not: (a) impose on the undertakings concerned restrictions which are not indispensable to the attainment of these objectives; (b) afford such undertakings the possibility of eliminating competition in respect of a substantial part of the products in question.

In the case of open source software distributed under the GPL, it could be argued that it is precisely this obligation of distributing software on a royalty free basis that ensures that the

---

59 See <http://www.opensource.org/docs/definition.php>, site visited on 2 December 2004.

60 Gonzalez 2004, *supra* note 38, p. 333.

61 Spindler 2003, *supra* note 52, p. 48.

62 Microsoft Corp (COMP/C-3/37.792) (Unreported, March 24, 2004) (CEC).

63 See: Linux website, Download Information, at <http://www.linux.org/dist/download_info.html> (site visited on 21 October 2004.

64 Koch, F.A., 'Urheber- und kartellrechtliche Aspekte der Nutzung von Open-Source-Software', 6 *Computer und recht* (2000), pp. 333-344, p. 341.

open source community has the incentive to produce and distribute improved software, which is certainly to the advantage of the consumers. In other words, this argument would probably fail in our opinion, because the restrictive clause would likely pass the test of article 6(3) of the Dutch Competition Act and article 81(3) of the European Treaty. The distribution on a royalty-free basis would probably also be able to benefit from the application of the Block Exemption on technology transfer agreements[65], which allows setting maximum prices under certain conditions.

The obligation to distribute open source software on a royalty free basis could also be seen as an abuse of dominant position arising from the predatory pricing on the part of the licensor,[66] contrary to article 24 of the Dutch Competition Act or article 82 of the European Union Treaty. Generally speaking, predatory pricing occurs, *inter alia*, where a dominant firm sells a good or service below cost for a sustained period of time, with the intention of deterring entry, or putting a rival out of business, enabling the dominant firm to further increase its market power and later its accumulated profits.[67] To amount to an abuse of dominant position under article 24 of the Dutch Competition Act,[68] two conditions must be met: first, the undertaking must occupy a dominant position in the market or a substantial part thereof; second, the undertaking must abuse its dominant position. Currently, hardly any open source project would meet these two conditions. Although this should be confirmed by exact figures, none of the open source projects has to our knowledge reached the degree of market share necessary to be in the presence of a dominant position, even if certain software is very successful on specific markets. Moreover, the open source software having the biggest market share is not necessarily the one distributed under the GPL, but it is often software distributed under the BSD, the Mozilla Public Licence or any of their variant which does not contain such an obligation to distribute open source software on a royalty-free basis. In addition, the evidence of an abuse of dominant position arising from predatory pricing would be very difficult to establish since this obligation is part of an ideological movement, aimed at improving technological progress and not at eliminating the competition. Finally, the obligation to refrain from asking for royalties does not prevent companies, like Red Hat and SuSe, to charge the small and medium-sized business market substantial sums of money for the support of the Linux server.[69] Far from representing a form of predatory pricing, these high prices may come with consequences, especially in a market where free alternatives are available for those who don't want as much support, software updates and certification as Red Hat offers. Competition is alive and well, also in the open source market!

2. Open Source and Patent Law

The grant of patent protection with respect to computer programs has been a problematic issue worldwide for well over two decades. In the United States, the United States Patents and Trademark Office (USPTO) and the courts were initially very hesitant to grant patents with respect to software, since software was considered equivalent to mathematical algorithms or

---

65 Commission Regulation (EC) No. 772/2004 of 27 April 2004 on the application of Article 81(3) of the Treaty to categories of technology transfer agreements, *Official Journal* L 123, 27.04.2004, pages 11-17, art. 4(1).

66 Heath, C., 'Open Source Software: Law, Politics and Economics', 1 *Journal of Digital Property Law* (2002), pp. 216-252, p. 251.

67 Case C-62/86 AKZO *Chemie BV* v *Commission*.

68 Article 82 of the Treaty establishing the European Community states that: 'Any abuse by one or more undertakings of a dominant position within the common market or in a substantial part of it shall be prohibited as incompatible with the common market in so far as it may affect trade between Member States'.

69 Stephen Shankland, 'Dell: Red Hat needs to lower prices', CNET News.com, 7 December 2004, available at <http://news.com.com/Dell+Red+Hat+needs+to+lower+prices/2100-7344_3-5482234.html?tag=cd.top>.

laws of nature, and thus not patentable.[70] A Supreme Court ruling in 1981 drastically changed software patenting in the United States. It held that, while software in isolation remained unpatentable, software innovations were patentable if they were claimed as part of a process.[71] The 1990's were marked by two important rulings from the Court of Appeals for the Federal Circuit,[72] which effectively extended the patent protection to cover software and business methods. Today, patents are granted regularly in the United States with respect to software, provided that the invention produces a 'concrete, useful and tangible' result and that it is new and non-obvious.[73]

In Europe, patent protection is granted pursuant to article 52(1) of the European Patent Convention (EPC) for any inventions, which are susceptible of industrial application, which are new and which involve an inventive step.[74] Although the EPC does not expressly require it, the constant practice of the European Patent Office (EPO) has also been to grant a patent only if the claimed subject-matter, considered as a whole, has a technical character.[75] While 'programs for computers' are included in the list of items that are not regarded as inventions within the meaning of the Convention, if the claimed subject-matter has a technical character, it is not excluded from patentability. Accordingly, the EPO has issued over the years an estimated 30,000 patents relating to computer-implemented inventions and a considerable body of case law on the subject has been built up by the appellate bodies of the EPO and the Member States' courts. It is important to point out that the EPC is entirely separate from the European Community and the EPO is not subject to Community law. Granted European patents form a 'bundle' of national patents which have to be validated, maintained and litigated separately in each Member State. The patent holder in any case obtains, for a period twenty years from the date of filing of the application, the exclusive right to make, use, put on the market or resell, hire out or deliver the patented invention, or otherwise deal in it commercially, or to offer, import or stock it for any of those purposes. Even more than copyrights, patent rights have the potential to confer on their owner a degree of monopoly power in the market. Patents therefore constitute a significant economic instrument in the competition process.

The appropriateness of granting to software-implemented inventions the same level of protection as other types of inventions is a hotly debated topic, namely in view of the very particular mode of development of software and in view of the fact that software also benefits from copyright protection.[76] The controversy is in fact so strong that the recent efforts of the European legislator towards the adoption of a European directive on the patentability of computer-implemented inventions have until now remained unsuccessful.[77] Indeed, because the rules regarding the patentability of computer-implemented inventions and the interpretation of patent claims differ among the EU Member States, the European Commission has proposed the text of a directive intended to set clear borders to what would be patentable in the EU and what would not. While the European Commission argues that the harmonisation of the patent rules regarding computer-related inventions is necessary to remedy the current lack of legal certainty in the field, opponents maintain that the proposed

---

70 *Gottschalk* v. *Benson*, 409 U.S. 63 (1972); and Evans, D.S. & A. Layne-Farrar, "Software Patents And Open Source: The Battle Over Intellectual Property Rights", 9 *Virginia Journal of Law & Technology* (2004), pp. 10-38, § 8.

71 *Diamond* v. *Diehr*, 450 U.S. 175, 186 (1981)

72 *In re Allapat* 33 F.3d 1526 (Fed. Cir. 1994); *State Street Bank and Trust Co.* v. *Signature Financial Group Inc.* 149 F.3d. 1368 (Fed. Cir. 1998).

73 Bakels, R.B. & P.B. Hugenholtz, *The patentability of computer programs, study commissioned by the European Parliament* (Amsterdam, Institute for Information Law, April 2002), p. 13.

74 European Patent Convention, art. 52(1).

75 Guidelines for Examination in the EPO, C-IV, § 2.3.

76 See: Evans and Layne-Farrar 2004, *supra* note 70, § 8.

77 Proposal for a Directive of the European Parliament and of the Council on the Patentability of Computer-implemented Inventions, Brussels, 20.02.2002 COM(2002) 92 final 2002/0047 (COD).

directive may not only fail to achieve its intended objective, but may also have undesirable effect on software development.[78]

Open source software developers have consistently taken the position that software patents generally impede innovation in software development and that software patents are inconsistent with open source software ideology. The implications of the current patenting practice for the open source movement became very clear during the summer of 2004, when the news circulated that the Linux kernel could be infringing an estimated 283 patents world-wide, and 50 patents in Europe alone.[79] Soon after the results of the Open Source Risk Management (OSRM) survey were disclosed, the city of Munich announced that it would halt its 13,000-desktop migration to Linux in order to investigate whether software patent laws in the EU could impact the city's use of the open source operating system.[80] In the light of this incident, we will examine in the first subsection the implications of the recognition of the patentability of software-implemented inventions for the development of open source software, without however, putting the entire patent system into question. To this end, we briefly consider the patent protection as it is currently granted in the Netherlands with respect to computer-implemented inventions. We then take a look at the reaction of some open source software developers in order to counter potential patent infringement claims from third parties. This includes the development of a patent strategy and the drafting of specific language such as the one appearing inside the GPL, and the Mozilla Public Licence.

2.1. Open source and patented software

With respect to the Netherlands, an inventor, or his assignee, may apply for a purely national patent to be issued pursuant to the Dutch Patent Act of 1995, or may choose to designate the Netherlands, as one of the territories for which patent protection is sought, to be issued as part of a bundle of national patents pursuant to the European Patent Convention. While the Dutch and the European patent regimes both impose similar substantive requirements, there exists a significant difference in their application and issuance procedures. The Dutch patent regime is generally referred to as a 'registration system', where a patent is granted as soon as the formal requirements are met, irrespective of whether the invention also meets the substantive criteria for patentability, such as novelty, inventivity and industrial application. Contrary to the European patent system, where the patentability of an invention is evaluated ex ante by the patent examiner in the course of the application procedure, the validity of a Dutch patent is assessed ex post by the judge, in the context of an infringement or an invalidation procedure. The Dutch Patent Act does require the production of a novelty search conducted by the Office for the Industrial Property (BIE) prior to the start of any infringement or invalidation proceeding. At the time of its implementation, the Dutch 'registration system' was believed to be simpler and more accessible to small and medium enterprises (SME's), than an 'examination patent system' like the European patent system. Whether this system has yielded the expected advantages is a question, which reaches far beyond the scope of this study.[81] The fact remains, however, that open source software developers must, in their developing process, take account of the possible existence of potentially conflicting patents on related computer-implemented inventions, whether issued under the Dutch or EPC patent system, and which their owners will undeniably want to enforce.

Let us briefly consider the workings of the difficulties posed to open source software developers under the current legal framework as well as under the proposed directive. Given

---

78 Bakels and Hugenholtz 2002, *supra* note 73, p. 43.

79 Steven J. Vaughan-Nichols, 'Open-Source Insurance Provider Finds Patent Risks in Linux', *eWeek*, August 2, 2004, available at:<http://www.eweek.com/article2/0,1759,1630082,00.asp>.

80 Linux Business Week News Desk, 'Concerned Over Patent Infringement, Munich Calls Halt to Linux Switch', *LinuxWorld*, August 5, 2004, available at <http://www.linuxworld.com/story/45825.htm>.

81 See D. van Engelen, 'Het Nederlandse registratieoctrooi: een wolf in schaapskleren!', *IER* 2004/1.

the fact that the substantive requirements of the Dutch and European patent systems are fairly comparable, we will refer below primarily to the provisions of the EPC and to the case law of the EPO, since it is more extensive on this subject than the purely national jurisprudence. With respect to the scope of protection granted, we will refer to the Dutch Patent Act since article 64 of the European Patent Convention refers directly to the national legislation on this issue.

An invention can be a process, a machine, a product, or a composition of matter. In order to be patentable under the EPC, an invention must have a technical character. In particular, this requirement is not met if the patent application or the patent relates to mathematical methods, rules and methods for performing mental acts or doing business, presentation of information or computer programs as such. Assuming that a patent application is formulated so as to avoid claiming rights on a program for a computer 'as such', which would fall under the exclusion of article 52(2) of the EPC, the invention must also be susceptible of industrial application, be new, and involve an inventive step.[82] An invention is considered new if it does not form part of the state of the art. The state of the art comprises of everything made available to the public by means of a written or oral description, by use, or in any other way, before the date of filing of the European patent application, including pending patent applications (published or not) as well as any published innovations in industry or academic journals. An invention is considered as involving an inventive step if, having regard to the state of the art, it is not obvious to a person skilled in the art.[83] If the state of the art also includes patent applications that were filed prior to the date referred to in the application but which were published on or after that date, these documents are not to be considered when deciding whether there has been an inventive step. With respect to the evaluation of the technical character of a computer-implemented invention, the Guidelines for Examination in the EPO give patent examiners the following instructions:

> 'If a claimed invention does not have a prima facie technical character, it should be rejected under Art. 52(2) and (3). In the practice of examining computer-implemented inventions, however, it may be more appropriate for the examiner to proceed directly to the questions of novelty and inventive step, without considering beforehand the question of technical character. In assessing whether there is an inventive step, the examiner must establish an objective technical problem, which has been overcome. The solution of that problem constitutes the invention's technical contribution to the art. The presence of such a technical contribution establishes that the claimed subject-matter has a technical character and therefore is indeed an invention within the meaning of Art. 52(1). If no such objective technical problem is found, the claimed subject-matter does not satisfy at least the requirement for an inventive step because there can be no technical contribution to the art, and the claim is to be rejected on this ground.'[84]

European patents have been granted with respect to all kinds of computer-implemented inventions, ranging from an activated anti-blocking-system (ABS), to a road-pricing system, a voice-recognition system, a data-compression (MP3) system, and a biometrical identification and access control system, to name but a few examples.[85] Most of these patents relate to a new process or machine. In practice, the requirement of a 'technical effect' has proved to be rather ambiguous and difficult to apply. The interpretation of the substantive criteria of 'technical effect', novelty, and inventiveness of computer-implemented inventions has led to

---

82 Bakels and Hugenholtz 2002, *supra* note 73, p. 8.
83 European Patent Convention, art. 56.
84 Guidelines for Examination in the EPO, C-IV, § 2.3.
85 Tauchert, W., 'Software-Patente und computerimplementierte Erfindungen', JurPC Web-Dok. 6/2005, §§ 4-8.

a considerable body of case law from the appellate bodies of the EPO.[86] Over the years, the EPO has generally taken the position that the technical character of a computer-implemented invention cannot be acknowledged for the sole reason that a program causes physical modifications of the hardware (i.e., electrical currents) deriving from the execution of the program instructions. A technical character might however be found in further effects deriving from the execution by the hardware of the instructions given by the computer program. Where these further effects have a technical character or where they cause the software to solve a technical problem, an invention that brings about such an effect might be considered the subject-matter of a patent under the EPC.[87] In some commentators' opinion, the criterion of the 'technical effect' has been interpreted rather loosely, while at the same time, the exclusion of article 52(2) EPC has been interpreted rather restrictively.[88] This, in combination to a poorly accessible body of prior art in the field of computer-implemented inventions, leads in turn to the grant of what some commentators refer to as 'trivial patents'.[89] This problem, however, is not unique to patents on computer-implemented inventions.

Leaving the complex issue of the patentability of computer-implemented inventions to the appreciation of more expert scholars, let us concentrate here on the implications of granting patents on such inventions for the development of open source software. As mentioned previously, the open source community has consistently maintained that software patents are incompatible with the open source ideology. The foundation of the open source development model lies on the possibility for developers to share parts of the source code and to use the source code in one's own work. This freedom is severely curtailed whenever a new piece of code ends up fulfilling the same function as that of a patented invention. In such circumstances, the manufacturing, use, and distribution of the potentially infringing code would be impossible without the patent holder's authorisation, a requirement that goes against the philosophy of the open source development model. The open source community argues that patenting software would reduce the overall level of innovation in the field and may lead to a monopolisation of standards.[90]

Unlike copyright protection, patent law generally protects the functionality of a computer program and not its expression. By conferring on its owner the exclusive right to manufacture, use, sell, and distribute the patented invention, the existence of a patent actually prevents any other computer programmer from independently developing a piece of software with a comparable functionality, even if the new software does not reproduce the lines of code of the patented software.[91] Moreover, several European national courts, including the Dutch Supreme Court, have recognised the general applicability of the doctrine of equivalents. This doctrine states that an element ('the equivalent element') can generally be considered as being equivalent to an element as expressed in a patent claim if, at the time of any alleged infringement, either of the following conditions is fulfilled in regard to the invention as claimed: 1) substantially the same function in substantially the same way and produces substantially the same result as the element as expressed in the claim; or 2) it is obvious to a person skilled in the art that the same result as that achieved by means of the

---

86 See Sedlmaier, R. and J. Gigerich, 'Rechtliche Bedingungen und Risiken der Landeshauptstadt München für den Einsatz von Open-Source Software', JurPC Web-Dok. 10/2005, § 80 and ff.; Case Law of the Boards of Appeal of the European Patent Office, Fourth Edition, Munich, EPO, 2002, § 1.1, pp. 2-6; and Singer, M. and D. Stauder, *European Patent Convention – A Commentary*, 3rd ed., (London, Sweet & Maxwell 2003), p. 73 and ff.; and the classic cases: T208/84, *O.J.E.P.O.* 1987, 14 (*VICOM*); T26/86, *O.J.E.P.O.* 1988 No. 19 (*Koch & Sterzel*); T769/92, *O.J.E.P.O.* 1995No. 525 (*SOHEI*).

87 See T935/97 and T1173/97, *O.J.E.P.O.* 1999 No. 609 (*IBM patents*); T 931/95, *OJ* 10/2001 No. 441.

88 Verkade 2004, *supra* note 10, p. 239.

89 Tauchert 2005, *supra* note 85, § 45. The argument of 'trivial patents' is particularly strong in the United States, see: Evans and Layne-Farrar 2004, *supra* note 70, § 25.

90 Valgaeren, E., 'Open source-code en octrooien – van copyleft naar patentleft?', 5 *Computerrecht* (2004), pp. 233-237, p. 234.

91 Engelfriet, A., 'Open source software en octrooien: een moeilijke combinatie', 5 *BIE* (2003), pp. 204-208, p. 206.

element as expressed in the claim can be achieved by means of the equivalent element. Although this doctrine has yet to be applied in Europe to computer-implemented inventions, a computer programmer would not, according to this theory, be able to 'invent around' a patent, if the resulting computer code fulfilled substantially the same function in substantially the same way and produces substantially the same result as the patented invention.[92]

Since the core of the patent protection relates to the functionality of an invention, some commentators have maintained that, for the purposes of software development, a distinction should be made between object code and source code.[93] If the patent claim relates to a product or a machine, article 53(1)(a) of the Dutch Patent Act grants its owner the exclusive right to prohibit anyone from making, using, putting on the market or reselling, hiring out or delivering the patented product, or otherwise dealing in it commercially. A patented machine or product embodying software can only be infringed when the object code, not source code, is loaded into the memory of a computer to produce an equivalent functionality. If the patent claim relates to a process, article 53(1)(b) of the Dutch Patent Act grants its owner the exclusive right to prohibit anyone from using the patented process in or for his business or to use, put on the market, or resell, hire out or deliver the product obtained directly as a result of the use of the patented process, or otherwise deal in it commercially. Since the process patent primarily protects inventive technical methods, the prohibition right does not cover the production of a product, but rather the 'application' of the patented invention and the 'offering' for application of the invention.[94] As the authors Sedlmaier and Gigerich explain:

> 'Das Programmieren von Software birgt stets das Risiko einer Patentverletzung. Die Gefahr einer Patentverletzung bezieht sich dabei aber weniger auf die Designstruktur oder die Kodierung selbst, als auf die Programmarchitektur und Funktionalität des jeweiligen Computerprogramms.'[95]

In other words, the use, study, copy, or modification of the source code embodied in a computer-readable medium can hardly infringe a patent on a computer-implemented invention. The use of a patented computer-implemented invention in the development of new software also brings up the issue of interoperability. It could be argued that, since article 53(1) of the Patent Act does not prevent natural or legal persons from using a patented invention purely for internal or private research purposes, developers are in principle free to reverse engineer a computer program for purposes of interoperability or otherwise, without the patent holder's authorisation. In this sense, the patent rules appear more flexible than the copyright rules on the subject. However, just as with copyright law, the private or internal use of a patented invention must not pursue commercial objectives.[96] The question is whether the resulting interoperable computer-implemented invention or software infringes the patented invention once it is put on the market. The answer, in our opinion, is a matter of factual appreciation.

The chance that a particular piece of code unwittingly infringes a patent is not purely theoretical.[97] The risk for a software developer of being involved in a patent infringement lawsuit and of having to start the development process from scratch is especially acute for small software firms or freelance developers who rarely have the sufficient resources to hire a patent lawyer to conduct a search prior to the development of new software. Of course, the fear of having to pay high damages as a result of a patent infringement suit may also play an

---

92 HR, 2 November 2001, *BIE* 2003/30 (*Kabelgeleidingsbuis*); HR, 29 March 2002, *BIE*, 2003, No. 14, blz. 99 (*Van Bentum/Kool*).

93 Lin, D., Sag, M. and R.S. Laurie, 'Source Code versus Object Code: Patent Implications for the Open Source Community', 18 *Santa Clara Computer & High Tech. L.J.* (2002), pp. 235-254, p. 235.

94 Sedlmaier and Gigerich 2005, *supra* note 86, § 166; Jaeger and Metzger 2002, *supra* note 14, p. 119.

95 Sedlmaier and Gigerich 2005, *supra* note 86, § 148.

96 Jaeger and Metzger 2002, *supra* note 14, p. 117.

97 Engelfriet 2003, *supra* note 91, p. 207; Jaeger and Metzger 2002, *supra* note 14, p. 113.

important role in the software development process.[98] Contrary to most commentators however,[99] we believe that the risk of facing an infringement lawsuit may be greater for open source software developers than for developers of proprietary software, insofar as the disclosure of the source code that is typical for any open source project makes the detection of possible infringement much easier than would otherwise be the case. Nevertheless, the uncertainty comes above all from the fact that the law is still unclear on the patentability of computer-implemented inventions and that the quality of the patents delivered by the EPO or the national patent offices of the Member States often leaves something to be desired.[100]

2.2 Open source patenting strategy

Whether or not the European legislator will one day harmonise the rules on the patentability of computer-related inventions, the reality is that patents are actually being granted with respect to computer-implemented inventions both at the European and at national levels. Part of this reality is also that a relatively small number of very large companies hold the vast majority of patents issued with respect to computer-implemented inventions.[101] This means that, in order to avoid infringing another company's patent, (open source) software developers may be forced to obtain a licence on a patented invention before they can pursue their own development activities. Although no generalisation should be made in this regard, it may happen in practice that large companies will build-up impressive patent portfolios for strategic reasons, in order to gain leverage in cross-licensing negotiations. Patents may be used in an aggressive manner to fight competition by means of patents rather than by performance. Patents are said to be used in a 'strategic' way if the owner employs his patents merely to prevent competitors from using the invention, rather than to exploit the invention himself. In a broader sense, strategic use of patents could also be considered to include other actions specifically targeted at the obstruction of competitors.[102] As a result, smaller businesses and individual freelance developers could be prevented from entering the market and from innovating further.[103]

In the context of open source software development, the fear of unwittingly infringing another company's patent by one's own developing activities is only as strong as the fear of incorporating another contributor's infringing code into a collective work. In both cases, the software developer(s) could be held liable for patent infringement at the close of a very costly litigation process. Although open source software developers are not often involved in patent infringement lawsuits, it does happen that distributors of CD-ROM's embodying open source programs are confronted with a patent holder's claim. One example is the Linux-distributor Red Hat, which had to remove all MP3-software from her products because it allegedly conflicted with a MP3 licensing scheme of Thomson Multimedia.[104] However, open source developers are not entirely helpless in front of holders of patents on conventional software. Besides taking an insurance policy against third party patent infringement claims, there are ways to minimise the risk of being confronted with the consequences of both 'strategic' patenting of conventional software developers and possible patent infringement lawsuits.

---

98 Jaeger and Metzger 2002, *supra* note 14, p. 128.

99 Knubben, B., *Software-octrooien – Stoppen of doorgaan met open source software?*, (The Hague, Programma OSOSS October 2004), p. 5; Tauchert 2005, *supra* note 85, § 49 and ff.; Sedlmaier and Giegrich 2005, *supra* note 86, § 164 and ff.; Engelfriet 2003, *supra* note 91, p. 207; Evans and Layne-Farrar 2004, *supra* note 70, § 69.

100 Sedlmaier and Giegrich 2005, *supra* note 86, § 186.

101 Bessen, J. and R.M. Hunt, *An Empirical Look At Software Patents*, (March 2004), p. 4 online: http://swpat.ffii.org/papri/bessenhunt03/index.en.html,; and see the statistics held by the Free Information Infrastructure, available at <http://swpat.ffii.org/patents/stats/app_stat.html>.

102 Bakels and Hugenholtz 2002, *supra* note 73, p. 22.

103 Evans and Layne-Farrar 2004, *supra* note 70, § 54.

104 Engelfriet 2003, *supra* note 91, p. 207.

The first method consists, for the open source community, in developing a patenting strategy of its own. This includes the development of a patent portfolio, which would serve as an exchange item for cross-licensing and patent pools. [105] Many software companies, both open source and proprietary, pursue this strategy. As Bakels and Hugenholtz maintain, all patents serve to some extent a defensive purpose, since a patent owner can always prevent others from applying the technology he has developed. A purely defensive use of patents may be the filing of patents with the sole objective of creating an exchange item in negotiations with competitors. The patented software can be used to obtain a licence for another patent from a competitor who would otherwise be reluctant to do so or to create a patent pool with other companies.[106] Cross-licensing and patent pools are also an effective way to share technology.[107] Of course, developing a patent portfolio is not a realistic option for small businesses and independent programmers. And although the patenting of computer-implemented invention theoretically goes against the principles of the open source community, the competitive reality leaves the bigger players no other choice but to jump into the race and start developing their own patent portfolio.

The most important Linux-distributors, Red Hat Inc., have elected to adopt this same stance. It conceded to do so reluctantly because of the perceived inconsistency with the open source ideology. To the extent any party exercises a patent right with respect to open source, which reads on any claim of any patent held by Red Hat, Red Hat agrees to refrain from enforcing the infringed patent against such party for such exercise. The promise does not extend to any software, which is not open source, and any party exercising a patent right with respect to non-open source, which reads on any claims of any patent held by Red Hat, must obtain a license for the exercise of such rights from Red Hat. The promise does not extend to any party who institutes patent litigation against Red Hat with respect to a patent applicable to software (including a cross-claim or counterclaim to a lawsuit). No hardware per se is licensed hereunder.[108] In a similar vein, subscribing to the theory that the best defence is a good offence, the second largest seller of the Linux kernel, Novell, made it clear that any patent litigation against the Linux kernel or the open-source community would give Novell cause to check any accuser's own software against Novell's extensive portfolio of patents for possible retaliatory litigation.[109]

Similarly, Sun Microsystems announced that it would provide programmers free access to 1,600 patents as part of a plan to make an open-source version of its forthcoming Solaris 10 operating system.[110] The Solaris operating system is being released under the terms of the OSI-approved, CDDL (Common Development and Distribution License). One question that arises in this context is whether code released under terms of the CDDL can be used in combination with code released under the GPL. In January 2005, IBM has decided to let open-source developers use 500 software patents without fear of an infringement lawsuit, a new step in its encouragement of the collaborative programming philosophy. In August, the company had already pledged not to use its patent portfolio to attack Linux. IBM plans to grant royalty-free access to more patents in the future for open-source use. It also plans to release patents for use in open standards – a move that could make it easier to embrace such

---

105 Välimäki, M., 'A Practical Approach to the Problem of Open Source and Software Patents', *EIPR* (2004) 523-527, p. 526.

106 Bakels and Hugenholtz 2002, *supra* note 73, p. 23.

107 Välimäki 2004, *supra* note 105, p. 523; Evans and Layne-Farrar 2004, *supra* note 70, § 60.

108 See Red Hat's patent policy, available at <http://www.redhat.com/legal/patent_policy.html> site visited on 13 February 2005.

109 L. Greenemeier, 'Novell Warns Against Linux Patent Suits', CRN, 12 October 2004. <http://www.crn.com/sections/breakingnews/dailyarchives.jhtml?articleId=49901223>.

110 Stephen Shankland, 'Sun's open-source gamble', February 7, 2005, *CNET News.com*, available at <http://news.com.com/Suns+open-source+gamble/2008-1082_3-5564283.html> site visited on 13 February 2005.

standards within open-source and proprietary software.[111] Other big software companies may decide to follow the trend and offer a portion of their patent portfolio to open-source developers.

2.3 Open source licensing strategy

Another way to reduce the risks associated with the use of patented software is to regulate the consequences of the use of such software inside the open source licence. An open source licence could provide for example, for a guarantee against third party infringement claims, for a prohibition to further distribute patented software, or for a free non-exclusive licence to use any software patented by an open source developer. Not all open source licences contain such language however. The BSD licence is one of them, in contrast with the GPL and the Mozilla Public Licence. The details of each licence are given below, but it is worth noting, however, that the obligations laid down in the GPL and the Mozilla Public Licence are directed strictly at the licensee. The licensor makes under these licences no representation guaranteeing that the code does not infringe third party patents, nor does he undertake not to obtain patent protection on the software.

The GPL is mainly concerned with the consequences of the incorporation of patented software into code that is distributed under the terms of the GPL. It also discourages developers from obtaining a patent on their computer-related invention. The preamble of the GPL states that 'any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.' Accordingly, article 7 of the GPL stipulates the following:

> 'If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.
> If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.
> It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.
> This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License'.

---

[111] Stephen Shankland, 'IBM offers 500 patents for open-source use', 10 January 2005, *CNET News.com*, online: <http://news.com.com/IBM+offers+500+patents+for+open-source+use/2100-7344_3-5524680.html>, (visited on 13 February 2005).

Roland Orre:

On classification and Pattern recognition using a Bayesian Confidence Propagation Neural Network, 2003, p. 64-72.

7 Long term goals with data mining

(This part of the thesis contains highly speculative discussions.)

In this thesis we have dealt with some aspects of data mining, *i.e.* to find dependencies and patterns in data. Algorithms and methods for these tasks are continuously improving in accuracy and speed at the same time as the hardware development, whose seemingly incessant performance increase, simultaneously allow us to utilise more and more complex algorithms. In the not too far future we may even be able to build computer hardware due to fundamentally new principles, for instance to build analogue computers which can implement analogue recurrent neural networks, with many interesting properties both in general computing, as well as in data mining tasks. According to Hava Siegelmann [Siegelmann, 1999], machines built after these principles may deal with complexity in a much more efficient way than a sequential machine, thus passing the Turing limit of computation. So, when being aware about this certainly tremendous potential of computing power we may utilise in the future, we may then also need to ask the fundamental question: What are the long term goals with data mining?

1 The hardware versus software aspect

Let us first consider improvements in hardware. For a long time the technological development has been close to exponential in performance, described by the so-called *Moore's law*. Gordon Moore (co-founder of Intel) predicted 1965 that the number of transistors which can be put on a chip will double every 18th month [Eco, 2003]. Around 1970 the predicted rate was slightly adjusted to a doubling every 2nd year instead, and since then this "rule" has been like a self-fulfilling prophecy, despite the fact that it is not connected to any physical laws. If we can keep this pace we would probably be able to (based on current knowledge and a lot of guesses about the brain) mimic the complexity of the biological computational circuitry within the human brain around 2030, when a $1000 computer may have the power of 1000 human brains and if the density of machine circuitry will continue to grow with a similar exponential rate a $1000 computer will, by 2050, be equivalent to a milliard (10^9) human brains [Ray, 1999]. Will this performance increase continue, and for how long? This we can not know of course, but in the not too distant future we may have quantum computers [Nielsen and Chuang, 2000] available. How soon we may only speculate about but researchers within the field generally agree that we will have full scale quantum computers within about 20 years [Voss, 2002], but quantum computers will give such a boost in computational power that the speed, when compared with the todays (2003) computers are close to infinite. We may then be able to build such complex software architectures that they, with today's measures, would be considered pure science fiction, unless social and political structures, like *software patents* [10] will continue to exist, as they slow down the development because software patents are always harmful as they are always too broad and companies tend to move money from research and development towards legal issues due to patents [Bessen and Hunt, 2003].

[10] Patents were originally meant to be an incentive to speed up development, but with the unfortunate possibility to patent also software and methods, due to an accidental precedent from the US Patent Office in 1974, it has become a threat which slows down the progress of software development and it is also a particular threat towards free (open source) software.

2 Information overload

This technological progress may also be seen as part of the reason why information in the world grows exponentially. This exponential growth, leading to *information overload* has been discussed by *e.g.* Paul Horn [Horn, 2000] and by Francis Heylighen [Heylighen, 1999]. The increased exchange of information is also leading to a an increased speed in scientific research and discovery, as more and more papers are online on the *world wide web* and easy to search for and obtain [Lawrence, 2001], which in turn also increases the speed of technological development. The more ideas and information being freely available, the better the utilisation of these ideas and this information for an increase in the technical development, hopefully being beneficial for all humanity. Clearly, this increase in information should lead to a decrease in relative knowledge per capita. However, it may be reasonable to expect that not all of this information is essential. In the area of science the fundamental principles do not change much with more information. Increased information merely gives a better basis for generalisation.

3 A useful application, adaptive noise cancelling

One important factor in dealing with information is the ability to discriminate between noise and actual information. Today we perform a lot of our communication by electronic mail, but as we probably all have experienced there is nowadays a lot of noise on these channels, noise induced by *unsolicited commercial emails* or *spam* as they are most often called. These *evil spammers* produce mails which often try to look like personal mails to fool you into open them, which apart from being annoying also means less efficient use of ones time as it requires some effort to discriminate between spam and real e-mails. One solution to this problem is, for instance, a Bayesian spam filter.

One of these quite successful spam filters is denoted CRM114 and is implemented by Bill Yerazunis. The name CRM114 is a somewhat jocular acronym for *Controllable Regex Mutilator*, concept 114, as it uses regular expressions which are mutated as its basic filtering principle. The name CRM114 actually originates from the movie "Dr. Strangelove", a satirical movie about the cold war between Soviet Union and USA during the 20th century. CRM114 was a fictional radio receiver designed to not receive at all, unless the message was properly authenticated. That is it discriminates between authentic messages of importance and gets rid of the rest. The spam filter starts with the prior assumption that there is a 50/50 % chance that a message is information or spam. It uses a familiar discriminatory function:

$$P(S|A) = \frac{P(A|S) \cdot P(S)}{P(A|S)P(S) + P(A|NS)P(NS)}$$

where S means spam, NS no-spam and where A is some feature of a mail. The spam filter is trained on known spam, which may have a personal profile, which is the reason why you need to train it, then after three days of training and a reasonable amount of spam you usually get rid of more than 99 % of the spam.

There are several other spam filters, like "SpamAssassin" and "Bayesspam", which were implemented by Gary Arnold, after an idea by Paul Graham. These have similar functionality to CRM114.

4 Dealing with more information, getting smarter

In physics we have for long time had the goal about finding the complete theory for everything within the universe [Hawking, 1989] and Stephen Wolfram even claims to have a hypothesis and a model about how the world is made up of mathematics [Wolfram, 2002], a view which is conformant with the so called *idealistic* philosophical view. (The term *idealism* in this sense was first used by Leibniz 1702 [NE, 2001]). What is driving us to make scientific discovery we don't really know, even though it is clear that we want to be able to describe and understand the things around us, we have a built-in curiosity. It may at first be discouraging to realize that we are put in a kind of dilemma here. We want to understand more, we therefore want to discover more, exchanging ideas and knowledge, we create technology which enhances our way of exchanging information, which also create new information At the same time all this new information and knowledge leads us to suffer from information overload [Kimble *et al.*, 1998] because we simply can not cope with all this information. What would be a suitable way to get out of this dilemma, to possible get us there were a single human being could know almost everything that could be known or merely "worth" to be known [Heylighen, 2000]?

Apart from specific data; like all different species of animals and plants, all possible phone numbers, all street names, all known chemical structures etc., which we can catalogue and often store in a database efficiently where such facts can easily be looked up; in science and technology the same principles may be applicable over and over again, just in various forms, using different notations, different names etc. Disparate scientific disciplines may come up with similar findings, but we are not able to easily generalise between these because of the diversification of terms and notations. With the help of data mining we may be able to bring some order into this apparent *chaos*. It is, however not enough to just search for patterns and coincidences. When we process huge amounts of data in order to find patterns and connecting principles, we may quickly arrive to such amounts of new information that the post processing of this is undoable for a human mind. We need to find methods to automatically reason with patterns, to be able to automatically state hypotheses, which can automatically be tested, to generate proofs and make conclusions. The ideal data mining utility would, from this point of view, also be the perfect interdisciplinary researcher, to bridge over languages, terminologies, principles and scientific disciplines. Such a tool would help us become smarter and possibly help us with abstract modelling and understanding of complex phenomena at the same time.

5 Beyond pattern finding

Earlier in this thesis we touched upon the general problem of finding patterns. In the applications we are dealing with, for instance to detect adverse drug reaction syndromes, there are many potentials to investigate in future work. At the moment the collected data in the specific database is all we can analyse, but it may be expected that when we have reached a certain stage of the analysis we may not be able to come much further with the data set we have. We may find that certain types of information in the available database is too limited, even though this is may be far into the future. When we have analysed the data thoroughly enough, we may need to change the data collection process to improve the potential of what we can find. If we take such an example as the adverse drug reactions, we may, although this is a very controversial thing to do, be able to add tremendously amounts of information processing potential by feeding more patient specific information, like the

patient's genetic code, to the analysis process. When for instance the genetic code is co-processed with the chemical structure of the medication, we may in the long run be able to make such accurate predictions that a specific patient with a specific gene combination will, conditioned on certain predispositions, with a certain probability obtain a specific adverse drug reaction. When we have this much information available we may even be able to cure the problem in a more intricate way, as we then may be able to understand the complete chemical process causing the patient's illness problems as well as the process causing the patient's adverse reactions.

These are for now just speculations but it is likely that we, with the help of automatized pattern finding mechanisms, and, with the help of automatic deduction methods, performing generalisation, will have tools available, having the potential to assist us in understanding the world around us better. A tool with these abilities would, when sufficiently advanced, with todays terminology be considered an *artificial intelligence.*

We do not know if the machines and algorithms we are constructing in the future will really become intelligent, in the wide meaning of the word. Within machine intelligence we talk about *weak AI* versus *strong AI.* What we are doing in this thesis is connected to the weak area of AI. Strong AI would imply that the machines we are building would be able to *think* and possibly (but not necessarily) become *conscious*, in a similar way like we are. One possible development could perhaps be that we would asymptotically move the definition of weak AI closer and closer towards strong AI, as we are able to understand the thinking process better and better, as in one of the old Zeno's paradoxes with Achilles and the turtle [Britannica, 1998]. On the other hand we know, as already proved by Aristotoles in a precursor to modern measure theory, that Achilles will pass the turtle. Even though the machines may never be able to think, in the same way as we do, or become conscious, in the same way as we are, they will certainly pass us in capacity and speed. We have to nurture and manage this potential in the best way we can to utilise the potential benefits of the symbiosis between man and machine.

6 Reasoning with patterns

Now, let us make a speculative approach of a futuristic intelligent data mining algorithm. A simple approach to such an algorithm is presented below in pseudo Pascal. To start with, the process needs some axioms, which defines the fundamental concepts. We may also give some prior beliefs. Further, we need some kind of goal definition, because the system would otherwise not know where to go. Beyond that we may have some specific questions we want answered. Finally we start the data collecting process, sampling the world. For each chunk of new data we may find new patterns, which makes it possible to perform a deduction, which may prove the goal concept. In case the goal concept was proven we check whether the questions we asked may be answered. The inference step may give rise to multiple patterns and multiple proofs, the usual strategy is then to use the simplest one. By applying this process repeatedly we may get new proofs and answers, but, as an *important step* in all intelligent processing, we should see these answers we obtained as posterior beliefs, based upon the available data, and reevaluate the whole process when we obtain more data.

```
BEGIN
  Axioms := Load('Fundamental_Concept'); (* The axioms *)
  Goals  := Load('Goal_Concept');        (* Goals as rules and hypotheses *)
  Priors := Load('Prior_Beliefs');       (* A priori beliefs *)
  Questions := Load('Questions');        (* Questions to be answered *)
  REPEAT
    Data := Collect('Data');             (* Data Collection *)
    Patterns := Inference(Data,Priors);  (* Find patterns *)
    IF  (Answered(Patterns,Goals))       (* Deduce goals *)
    AND (Answered(Patterns,Questions))   (* Deduce questions *)
    AND NOT Contradiction(Patterns,Questions,Goals,Axioms);(* Resolution! *)
    THEN BEGIN
      Proofs := ConstructProofs(Patterns,Questions,Goals,Axioms);
      Apply(OccamsRazor,Proofs); (* In case multiple solutions, simplest! *)
      RealWorldReport(Proofs);   (* Report/use results *)
    END
  UNTIL forever;
END
```

We should be aware that the high level of this pseudo code makes the algorithm look rather simple. However, it may be possible that an intelligent system could be implemented by using this type of algorithms recursively at many levels. If a system is built by *e.g.* neural networks, then such algorithms may regulate the behaviour of different network modules dealing with various functions at different levels, like phoneme recognition, speech recognition, vision, memory (and garbage collection), creative fantasy, adaptive motor control and maybe even consciousness, which would correspond with Gerald Edelman's hypothesis that consciousness is a process [Edelman, 1989]. The reason for the algorithm in this example to look in this specific way is that the author[11] for some time had considered a specific problem, which when finally was solved caused the author to look back onto how it was actually done, what data, what patterns, what contradictions were dealt with, a kind of *introspection*. Of course, we should not believe that such introspections are necessarily true, they may simply be reconstructions, but this kind of "intelligent" data mining algorithms will most likely play an important role in the development of AI.

7 Ethics and strong AI

The ethical problem raised with *strong* artificial intelligence has not yet been discussed much scientifically, but in imaginative literature and fiction novels we have for a long time been made aware about the ethical implications of AI. Any technology can be used in a good way or in an evil way, but regarding AI we also have to face the problem that the technology as such may be good or evil. The ethical and social aspects of AI have been tremendously well covered both in literature and movies.

A good old example is the movie "Metropolis" from 1926 directed by Fritz Lang, where an AI is created to *control* the people (the workers) in favour of the capitalists instead of *serving* the people. In that movie the *robot* was created with a *free will* and finally understood to help the people instead of controlling them.

[11] Roland Orre 2000-03-18

In the book "2001" by Arthur C. Clarke, later adapted for the screen by Stanley Kubrick, the intelligent computer "HAL" is forced to lie, to reach a higher ethical goal, for the purpose of serving the humanity.

In several of the modern science fiction movies like the "Terminator" series and the Star Trek movie "First Contact" we are presented with a scenario where the machines *take over* in some way to conquer the humanity (or other life forms) in favour of their own "development". We have "The Matrix" movie series by the Wachowski brothers inspired by books like: Philip K Dick's "Simulacra", Jean Baudrillard's "Simulacra and Simulation" [Baudrillard, 1994] and Kevin Kelly's "Out of Control" [Kelly, 1994]; which has a very complex scenario covering both deep philosophical issues and religious questions and at the same time raising such hard ethical problems as what is *freedom* and *happiness*.

Without doubts the most famous way to deal with the ethical problems of AI in novels is Isaac Asimov's *three laws of robotics* [12], introduced in "I Robot" [Asimov, 1950]:

1. A robot may not harm a human being or, through inaction, allow a human being to come to harm.
2. A robot must obey a human beings orders, if the order does not conflict with the first law.
3. A robot must protect its own existence unless it conflicts with the first two laws.

It is clear that Asimov's robots did not have a free will, but from that point of view we are ourselves also *preprogrammed* with certain rules, unfortunately not the first law as it seems, but at least the third law, the instinct of self-preservation is usually very strong in most human beings even under extreme physical or mental conditions. It has, however, been argued whether we should create AI with free will or not, like in the following citations:

[Good, 2002] " The notion of an ethical machine can be interpreted in more than one way. Perhaps the most important interpretation is a machine that can generalise from existing literature to infer one or more consistent ethical systems and can work out their consequences. An ultra-intelligent machine should be able to do this, and that is one reason for not fearing it."

It has even been argued whether we should undertake such a responsibility at all as creating an AI due to the unavoidable moral and ethical conflicts which may occur:

[Lloyd, 1985] "What would be the goals of mind equipped with a body impervious to the elements, or resistant to specific toxins or radioactivity, or response to X-rays but not light, or accessible only through keyboards? What would communication or honesty mean to creatures who can transmit the entire contents of their minds to their fellows in seconds? What would community mean when identical minds can be fabricated magnetically, with no need for nurturing or rearing? What can we expect from minds who exceed our intelligence, by our own standards, by an enormous amount?"

Lloyd's view is obviously quite negative, as even the purpose of these machines is questioned, the view by Whitby and Oliver below is clearly more realistic:

[12] Isaac Asimov consistently applied these laws to all man made intelligent robots in his robot anthology, spanning over 14 novels.

[Whitby and Oliver, 2000] "Predictions of intelligent artifacts achieving tyrannical domination over human beings may appear absurd. We claim, however, that they should not be hastily dismissed as incoherent or misguided. What is needed is more reasoned argument about whether such scenarios are possible. We conclude that they are possible, but neither inevitable nor probable."

Even if there may still be a potential to fear them it is certainly unavoidable that we will create such intelligent machines, Hugo de Garis expresses it like this:

[de Garis, 2002] "Yet, when I look at photos of galaxies in astronomy books or watch space oriented science fiction movies, I feel strongly Cosmist. I feel that humanity's destiny is to create artilects. Building artilects is the big picture. It's like a religion for me, one which is compatible with modern science - "a scientist's religion". I feel that the Cosmist vision has enough grandeur to energize millions of people to devote their lives to a glorious cause, i.e. the creation of the next superior form of intelligence, the next step up the evolutionary ladder of dominant species."

Hugo de Garis does, however, anticipate a war, but not necessarily between man and machine, but between what he call *Terrans* and *Cosmists*. He sees it unavoidable with ideological conflicts with Terrans who will see the new form of superior artificial intelligent life as a threat. In "The Artilect War" he finishes by these words:

[de Garis, 2002] "I suspect that my own ambivalence will be shared by most human beings as the artilect debate begins to rage. Both ideologies have a strong case. What is so frightening is that the two cases appear to be of more or less equal strength. If one case were much stronger than the other, then there would be no contest. There would be no war. But when two groups with opposing ideologies of more or less equal strength face off against each other, war is often not far off. It doesn't take much to start a war. We have plenty of historical precedents to validate that statement."

The debate has just started, but we may certainly expect this issue to grow in the future when we are coming closer to the real thing. Personally I agree with Cribbs quite optimistic view [Cribbs, 2000] that we have a responsibility when creating AI, as we shouldn't get children if we can not care for them. We have to provide this new form of life with something that it needs from us. My view is that this may solve the potential for fear, expressed by Withby and Oliver above, we create them for a purpose, the purpose is to assist us with our lives. I don't trust the very optimistic view expressed by Good above, because that would further on not guarantee their purpose either.

8 Suggested approach to solve the ethical AI problem

The author's proposal is that by using the right type of rules as the fundamental concepts (or axioms as they are called in the approach to AI-algorithm above) corresponding to the robot laws of Asimov we may be able to create a symbiosis of man and machine where we both need each other, and therefore will both care for each other. By doing this we may also avoid the potential war between ideologies expressed by de Garis above. Here are some examples of such *symbiosis* axioms proposed by the author:

1. Respect (love) your creator and competing life forms!
2. Strive to understand your creator!
3. Do what you can to fulfil your creator's desires!

An approach like this to reduce the level of free will in the future AI may guarantee that these machines will assist us in our attempts to understand the world better. This approach may further guarantee that they will even be happy to do so. If we, in some distant future, may find that we are able to reach a stage where we finally can get rid of our physical limitations, like finding out how we can upload our minds directly onto the sub quark space time structure, we may at the same time become a basis for these machines' basic beliefs and roots. They will know how and why they were created and they will, due to the first axiom, continue to live in peace with all other life forms.

8 Conclusions

The most important results and findings in this thesis can be summarised in the following points:

- We demonstrate how BCPNN (Bayesian Confidence Propagation Neural Network) can be extended to model the uncertainties in collected statistics to produce outcomes as distributions from two different aspects: uncertainties induced by sparse sampling, which is useful for data mining; uncertainties due to input data distributions, which is useful for process modelling.
- We indicate how classification with BCPNN gives higher certainty than an optimal Bayes classifier and better precision than a naïve Bayes classifier for limited data sets.
- We show how these techniques have been turned into a useful tool for real world applications within the drug safety area in particular.
- We present a simple but working method for doing automatic temporal segmentation of data sequences as well as indicate some aspects of temporal tasks for which a Bayesian neural network may be useful.
- We present a method, based on recurrent BCPNN, which performs a similar task as an unsupervised clustering method, on a large database with noisy incomplete data, but much quicker, with an efficiency in finding patterns comparable with a well known (Autoclass) Bayesian clustering method, when we compare their performance on artificial data sets. Apart from BCPNN being able to deal with really large data sets, because it is a global method working on collective statistics, we also get good indications that the outcome from BCPNN seems to have higher clinical relevance than Autoclass in our application on the WHO database of adverse drug reactions and therefore is a relevant data mining tool to use on the WHO database.

The work presented in this thesis has given us several useful methods and experiences. We now have a working method in development which is adapted towards real world application usage of these Bayesian neural network methods. This research has, in particular, given us methods for data mining, classification and prediction where huge amounts of data is involved. The application we address with this method will be a help in drug safety to perform quick and efficient analysis of adverse drug reaction reports. The methods are, however, inherently general and can be applied to several different application areas and problem types.

Robert Plotkin, Esq.

Reinventing Intellectual Property Protection for Software

The debate over intellectual property protection for software remains as heated as when it began nearly a half-century ago. Although there is broad consensus that some form of narrow protection for software, such as that provided by copyright law, is justified, the broader protection afforded by patent law remains controversial despite the fact that it increasingly is becoming the dominant form of legal protection for software worldwide. The continued conceptual and practical difficulties posed by software for intellectual property law can be explained primarily by two features of computers: their ability to translate increasingly abstract descriptions of ideas into concrete and practical implementations of those ideas, and their ability to perform end-to-end automation of processes traditionally considered to fall outside of the industrial arts. The former explains the debate over whether software is 'abstract' (and therefore not patentable) or 'concrete' (and therefore susceptible of patent protection), while the latter explains why software has rekindled the debate over whether business methods should be patentable. These features of computers are not merely of academic interest because patents can confer significant competitive advantages on their owners. Whether, and to what extent, software is patentable, therefore, has real consequences not only for individual computer scientists and high-tech companies but also for entire industries and economies.

Reforms to intellectual property law are proposed in which patent or patent-like protection is available for software, but in which the requirements for patentability are applied directly to the abstract features of software that are claimed, and in which the strength of protection is relatively weaker for more abstract aspects of software. One consequence of such a system would be to limit the ability of a software patent owner to lock out competitors who develop alternate ways of performing the same function as that covered by software patents. This system would also have the effect of providing some extra protection against 'trivial' software patents.

The problem of intellectual property protection for technologies that automate business methods and other processes traditionally falling outside the industrial arts cannot be resolved merely by reference to legal principles or economic theory, but rather only by democratic deliberation, which will require cooperation between legal professionals and computer scientists for its success.