Journal of Information, Law and Technology

Software Development, Intellectual Property, and IT Security

Robert A. Gehring

Assistant Lecturer
Technical University of Berlin
Department of Electrical Engineering and Computer Science
Computers and Society Research Group
rag@cs.tu-berlin.de <mailto:rag@cs.tu-berlin.de>

A preliminary version of this paper was presented at the international congress 'Innovations for an e-Society. Challenges for Technology Assessment', held October 17-19, 2001 at Berlin, Germany. The author wishes to thank patent attorney Axel H. Horns for his explanations of some of the peculiarities of the international patent regime.

This is a **refereed** article published on: 4 July 2003.

Citation: Gehring, R, 'Software Development, Intellectual Property, and IT Security', 2003 (1) *The Journal of Information, Law and Technology (JILT)*. <http://elj.warwick.ac.uk/jilt/03-1/gehring.html>

Abstract

This article presents the argument that insecurity of software is due to interaction of technological and legal shortcomings, fostered by economic rationality. Ineffective

liability laws further the distribution of unreliable and insecure software. Copyright protection for software hinders quality improvement. Patent protection encourages binary distribution of code and the use of proprietary instead of standard technology. Open source software development is proposed as a starting point for a risk management strategy to improve the situation. Existing patent laws should therefore be modified to include a ‘fair use' defence rule - the 'source code privilege' (Lutterbeck et al.; 2000). Finally, some of the pros and cons of introducing a 'source code privilege' are discussed.

Keywords: software engineering, information security, software reliability, copyright, patent law, intellectual property, risk management, liability, open source software, source code privilege

1. Introduction

There is an ongoing legislative process of enhancing intellectual property rights in the fields of copyright protection and patent protection. How the security of information technology is affected by laws and technical measures is rarely considered, at least if one takes as a guide the political and legal processes that lead to new legislation.1

Lobbyists of the right owners put heavy pressure on the politicians to enact laws to protect their commercial interests - often successfully. As the topic of security is approached, the right holders worry about the safety of their respective intellectual property assets but actually do not care much about the security of the underlying information infrastructure. 2-3

Consider the following example:

'Digital Rights Management (Security). You agree that in order to protect the integrity of content and software protected by digital rights management (‘Secure Content’), Microsoft may provide security related updates to the OS Components that will be automatically downloaded onto your computer. These security related updates may disable your ability to copy and/or play Secure Content and use other software on your computer’. 4

This notice is taken from the End User License Agreement (EULA) accompanying a so-called ‘security- patch' for Microsoft's Windows Media Player. 5 The users in fact are forced to hand over the control of their respective computers to a company that is well known for its insecure software. 6-7 Questions: Who else will also use this ‘back door'? And for what purpose? Answer: Nobody knows!

The latest intellectual property and contract laws back such questionable conduct. 8-9 The attitude embodied therein poses a critical problem for security in a networked world. Laws that protect insecure software are going to foster at the same time network insecurity. The more the institutions of society get electronically networked the more they expose themselves to risks -- from unbalanced laws.

1.1 Empirical Indications

Some empirical examples from the networked world will illustrate what risks we are

already confronted with.

- In 1999, the so-called 'I love you'-virus caused worldwide damages of about 12 billion dollars.

- That was more than twice the damage that was counted in 1998 due to all viruses together.

- In 2000, the same virus caused damages of not less than 6.7 billion dollars within 2 weeks (McAfee; 2001).

And viruses are only a part of the problem.

- The burden of dealing with unreliable software adds up to an estimated US$78 billion per year in the U.S. alone, industry leaders say (Levinson; 2001).

- A recent study (Thibodeau; 2002) of the U.S. National Institute of Standards and Technology (NIST) estimated an economical loss of about US$ 60 billion each year due to defective software: '[U]sers incurred 64% of the cost and developers 36%'.

These examples should be sufficient to demonstrate how serious the problem of unreliable and insecure software is. Unfortunately, there is no crisp definition of security and reliability of software. Thus it is hard to decide if the figures shown above give a correct picture.10 The scale of the problem, however, should become clear.

1.2 Terminology

For the purpose of the current discussion, a working definition of security and reliability is needed. 11 Following the traditional conception, security can be seen as the property of a system to resist unauthorized access to and use of the system's resources. 12-13 Reliability is the property of a system to do what it is supposed to do and not to do what it is not supposed to do.14 Security and reliability depend heavily on each other: A security breach often results in a reduction of reliability and, in turn, the lack of reliability often results in security risks.15 With respect to security and reliability, the focus of this investigation is set on software, unless stated otherwise. 16

2. Thesis

As the following analyses will show, the insecurity of software is not due to a conspiracy of fate. It is the result of what may be deemed a misregulation. The thesis this paper intends to support can be formulated as follows:

Insecurity of software is due to interaction of technological and legal shortcomings, fostered by economic rationality.

The three pillars this thesis is built upon will be examined in turn.

3. The First Pillar: Technical Shortcomings

The starting point will be a short examination of technical reasons embodied within the process of software development. The three distinguished steps of the software development process are (Viega and McGraw; 2001, p. 17):

1. Requirements design,
2. Detailed specification, and
3. Implementation.

In practice, however, not all three steps get the attention they deserve, as will be discussed in the following sections. 17

3.1. Incomplete Specifications

The software development process has inherent limitations. Software development is of a rather subjective nature. Compared to other fields of engineering, experience and intuition play a more important role. Software engineering is not formula-based.

The first and most important limitation of the software development process can bee seen in the incompleteness of functional specifications. The functional specification is derived from the analysis of the user's (customer's) requirements. It serves as the blueprint for the software and contains the instructions about what code the programmers will have to write.

Additionally, many test instructions (for the so called 'white-box' tests) are derived from the functional specification. Every 'hole' in the specification will have direct impact on the testing process. And there always are 'holes' in the specification because of 'formal specifications [...] are at least as difficult to create, and as error-prone, as programs' (Hamlet; 1995, p. 194).

Measured by real world conditions, a functional specification is sometimes -- and in practice most times -- incomplete with respect to the functional requirements. And it is nearly always incomplete when it comes to security considerations.18 Design for security is not part of the classical software development process! 19

3.2. Incomplete Testing

Software that is written in perfect compliance with its functional specification and has passed all the necessary tests is called correct. 20-21 But testing procedures are incomplete themselves. 22

To be sure, a lot of mistakes are detected during the testing procedures. But not all errors in a complex program can be found through testing. For a complex program, exhaustive testing is unfeasible due to constraints of time and budget. 23 Tests usually show only snapshots from the spectrum of the software's behaviour, not the whole picture. Thus, confidence in the quality of the code can only be based on statistical assumptions (Hamlet; 1995, p. 194). No serious programmer would claim that a certain piece of software is secure because it is tested to be correct. 24

Last but not least, testing has an ontological limitation of its own. Testing can only reveal errors present in the program. But it cannot show the absence of errors (Floyd and Züllighofen; 1997, p. 644 and Kaner; 1997a). 25 In short: You cannot show what is not there!

4. The Second Pillar: Economic Behaviour

Following this exposition of the software development process, we shall now consider the economic consequences of the weaknesses we identified. The two main parties involved in commercial transactions are software vendors and buyers.

4.1. Software Vendors

When discussing the behaviour of software vendors, the 'homo economicus' model of the social sciences is valuable. Software vendors are rational actors. 26 Making use of all information they are able to gather and process, they try to optimise their subjective utility. Subjective utility can be optimised by maximising market share and prices, and avoiding costs wherever possible.

4.1.1. Profit-seeking and Liability

Commercial off-the-shelf software is a product for a mass market. Clearly, a product is only of value to its seller when a buyer can be found. Prices, and hence profits, are limited by market demand. 27

However, testing and debugging software is expensive. Sophisticated testing and debugging will drive up the prices for software and cut profits. If the expected costs of liability for distributing insecure software in total are lower than the expected costs of a more complete testing and debugging process, to deliver an unsafe product will be preferred by the software producer. 28 Under these conditions, we may expect that the costs of software testing will be shifted to the customer. 29 The user ends up struggling with complex, unreliable technology. 30

4.1.2. Limited Service

Since the service capacity of a mass-market software producer is limited regarding the millions of customers, the amount of available service is limited. And scarce goods are pricy within a market economy. Only a minority of customers can afford to buy the necessary service.

4.1.3. Service as a Business Model

Looking at the other end of the software market, to the vendors of tailor-made software, the findings are not more encouraging. Their business model includes profits from selling service and support for unreliable products (Levinson; 2001). This is economic behaviour.

4.2. Software Buyers

The software market shows strong information asymmetries.

- The vendor knows more than his customer about the quality of the product before

selling it. 31

- The customer learns about the quality of software only after purchase, not having tried it out before. Certification, as it is often proposed to close the information gap, will not be successful under the existing regulative framework, as Anderson (2001b) explained.

- No objective metric for software quality exists. Consequently, '[e]ven if consumers are willing to pay for more secure systems, choosing a system based on its security properties is difficult. This is not a failing of the consumer, as even industry experts rarely have little more than crude heuristics available to them to compare the security of competing products' (Schechter; 2002, p. 1).

Hence, quality considerations cannot have a great influence on the customer's preferences. In the result, the customer's purchase decisions won't be based on security concerns but rather be motivated by the price of a product, or other criteria not related to security. That is the way a market with adverse selection, a so-called 'market for lemons' develops (Akerlof; 1970).

Within such markets there is no place for the better product and, consequently, the market fails to deliver security to the customers.

4.3. Network Externalities

Software is a so-called experience good within a market with strong network externalities. 32 The positive feedback from the network externalities usually works to the advantage of the largest supplier and to the disadvantage of smaller competitors. In the case of new technologies to be introduced, time to market may play a pivotal role (Hamlet; 1995, p. 194).

Incentives to be first on the market and to establish one's own products as de facto standards are very high. 33 If lowering security precautions and using proprietary technology gives a competitive advantage, (dominant) market players will do so in order to preserve and expand their market share. Hence, considerations of establishing reliability and security lose priority against luring customers with new and more features.

Once there is a dominating market player, users are forced to deploy its products to be able to communicate with other users. Customers have little choice but to grasp the solution that gives them interoperability. 34

5. The Third Pillar: Legal Obstacles

The thesis about the conditions for insecure software is built on a third pillar: the legal obstacles to software quality improvement. The obstacles mainly can be found in three different areas of the law.

1. Liability rules, and

2. Trade secret laws, and
3. Intellectual property laws.

Liability laws are identified as a hindrance because they either do not exist or, if they exist, do not fit easily the characteristics of modern software technology. Lack of liability means for a software vendor that he can put more money into the production of new (IPR-protected) software instead of investing into the correction of existing products. This perpetuates the 'vicious cycle' of producing inferior software. 35

Within the constraints of the current discussion, we cannot discuss the nature of liability failures in detail. 36 It is sufficient to assume that modern, complex software cannot be produced without bugs and weaknesses. Some of the reasons were discussed above. It is evident that all software on the market contains errors. We should keep this in mind.

The next sections shall focus on the third legal area: intellectual property rights laws (IPRs). Though trade secrets by their very nature are not intellectual property rights, regarding software, trade secret protection and IPR-protection are tightly interwoven 37. Trade secret protection allows for a short span of lead-time and thereby works similarly to weaken IPR-protection (Samuelson and Scotchmer; 2002, p. 1586). We begin by highlighting copyright shortcomings, followed by a section pointing out some shortcomings of patent law and their unintended consequences.

5.1. Copyright Shortcomings

Copyright law gives the exclusive right to modify the code of a program and to distribute modified programs to the holder of the copyright. Acknowledging the public interest in the availability of a broad range of software, there are some minor exemptions from these exclusive rights.

Copyright protection for software contains a broad ban - with a few exceptions - on reverse engineering. In general, reverse engineering is only allowed for matters of operability and interoperability. 38-40 These exemptions are insufficient from a security point of view:

- There is no exception for security evaluation and/or enhancement.
- There is no exception for removing minor errors that influence the behaviour of other programs.
- Self-help of software users regularly is declared an illegal activity (Rosenoer; 1997, p. 22-26).

For copyright law, IT security is almost completely out of focus.

5.2. Shortcomings of Patent Protection

The second important hindrance, within the realm of intellectual property protection, to the enhancement of software quality and security has to be seen in the increasing patent

protection for software.

Patent law gives the patent holders the exclusive rights to the patented technology - in every possible implementation. Protected are the functional elements of the patented technology. It is important to notice that due to the broad protection the patent law gives, there are hardly compatible substitutes (second sources) for a certain technology. 41

Three points show the security problems connected to patent protection for software:

(1) Binary distribution of software will be preferred because of two advantages to the supplier:

- Possible patent infringement that cannot be avoided can be hidden thereby, and
- Trade secret protection is sustained (Samuelson and Scotchmer; 2002, p. 1608). 42

Through binary distribution of software, customers are confronted with an opaque machine they have to rely upon. Thus, transparency with respect to the quality of code is diminished and the market for ‘lemon-software' is fostered. 43 Other obstacles due to patent protection include:

(2) Secure technology itself may be patented. Thereby the fast spread of secure technology is delayed. 44

(3) Business models with a core idea of securing systems may be patented. This is already happening. 45

The outcome of the last two points is that one has to acquire a licence in order to make a system secure or to fix security flaws in a certain way. If transaction costs for negotiating contracts for licences are too high for the software developer, no agreements will be made, as we have learned from the anticommons theory (Heller and Eisenberg; 1998 and Heller; 1998). Since complex software contains a high quantity of know-how worthy of patent protection under modern patent doctrine, negotiation costs are arguably much too high for most software developers. 46 This contributes to the low level of IT security of current software products.

5.3. Preliminary Summary

Software is a special matter. Software may be characterised as a ‘functional texts'. By its very nature, software is simultaneously describing as well as ‘realizing' conceptions of parts of the world. Cognitive constraints and economic demands make it unlikely, indeed almost impossible, to write perfect software. Software is imperfect, i.e. it tends to be unreliable and insecure. Nevertheless, its deployment is prevalent in modern life, confronting us with a need to deal with its imperfections.

Our relationships rely more and more on electronic communication, i.e. on exchanging information encoded as data and transferred between different types of software. The interoperability of the software at both ends of the communication channel is decisive for

a successful communication. To take part in such a networked communication you have to have the ‘right' software. If you try to escape this pressure of conformity, you may end up with broken relationships. Described as a ‘network effect', this phenomenon explains why people decide to make their purchase decisions on the basis of compatibility considerations rather than product quality.

Intellectual property laws were not drafted with ‘functional texts' in mind. Software blurs the borderline between copyright protection (affecting elements of expression) and patent protection (affecting functional elements). Both copyright protection and patent protection come with a set of disadvantages that may be tolerable as long as they are not combined. For software, the interaction of copyright and patent law leads to business decisions disregarding product quality.

Normally, there are two levers for regulating ignorant behaviour: liability rules and market forces. In the case of software, both levers are broken. Effective liability rules do not exist nor can ‘the invisible hand of the market' guide decisions. The exclusive rights of intellectual property law protect even irresponsible actors from competition.

Taken together, these factors may explain why unreliable and insecure software is dominating the market. The next section will consider ways to improve this unsatisfactory situation.

6. How to Move Forward?

In view of the error-prone development process described above leading to faulty software, which in turn leads to insecure systems, we should develop an appropriate risk management strategy. Such a risk management strategy has to be devised in a manner that it will reduce the risks associated with the use of software over time without introducing new risks.

The best method known for enhancing the quality of software is the deployment of well-tested standard components combined with an extensive process of peer review. 47 However, we need a better peer review process than that provided by a single software producer both in terms of performance and in terms of independence. It must be scalable to the magnitude of the ever more ubiquitous Internet; it has to have short response time cycles; and it must be transparent. Transparency is of paramount importance in order to overcome the adverse selection problems. To current knowledge, there is only one software development process that fulfils these requirements and at the same time supplies the basis of a security process - the open source software development model. 48

6.1. The Strengths of the Open Source Approach

In this section, some of the key features of the open source model are discussed, influencing the reliability and security of the distributed code. They have to be seen mainly in the field of bug/incident management, user relationship management, and software licensing.

The open source software development process is based on the unrestricted availability of

the software's source code - not only to the producer but also to the users of a certain piece of software. 49-50 Together with the source code come licences that give to the users a non-exclusive right to modify the software and redistribute modified code. We can think of this combination of code and licences as of the establishment of a digital commons. 51

Users all over the world take the code basis, run, test, enhance and secure it. And they give the enhancements back to the developers and other users. That helps to drive down the dead-weight loss resulting from independent individual investments in software security occurring in a perfectly competitive environment. Managing the digital commons through allowed sharings and required givings contributes to avoid a tragedy of the digital commons, as well as unintentionally bringing about an anticommons situation as described by Heller (1998). 52-53 The software process as a whole gains from the enhanced feedback cycle (compared to the one deployed by commercial software vendors) in a specific way: Earlier stages of the final product are tested with a test set much more representative for real world conditions than the artificially chosen samples in the classical software process. As long as at least some of the detected errors in the code are corrected, not only the software process but also the final software product is improved. 54

Time is crucial in the case of security relevant incidents. This is where open source code gives an advantage. Only open source code enables users at work (and at home) to fix security weaknesses as soon as they are detected. 55-56 And fast and appropriate response is one of the cornerstones of a good risk management strategy, beside prevention and detection.

With open source software, users themselves become part of the software development process. 57 Evidently, that shifts power away from the supplier of the software to the users. At the same time, competition and user's choice is fostered. Openness of interfaces allows for the supply of compatible and complementary products. Thereby, the position of dominant market players erodes which may well force them to raise their quality efforts. Otherwise, they may lose out to competition in the long run.

Since the source code of the program is publicly available, there is no need for reverse engineering before being able to understand how it works. This is not the solution but the decisive prerequisite a number of security experts demand - or recommend - in order to make secure systems available. 58

Following these arguments, one can say that the open source model at least deserves a chance to prove its promises on delivering better software. But this model is in danger. The gold rush in software patenting reflected in a steep growth in patent applications and grants, in particular in the United States, may well stall what looks yet so promising.

6.2. The Patent Threat

Regarding the open source software development process, there are serious problems accompanying patent protection for software. Open source software developers are particularly vulnerable to patent litigation because the code of the programs is open to

everyone to inspect for patent infringement. On the other hand, the vendors of proprietary software, which distribute their products in binary form, are widely protected against such investigations by the ban of reverse engineering in copyright laws.

In recent years, the uninhibited patenting of many trivial idea if only expressed in patent lawyers' terms and implemented in software, has led to the much criticized situation that complex software normally should not be written any longer without extensive patent search. However, this is a job for a patent specialist, not for the average programmer. Large software producers do have a patent department at their hands. Typical open source developers, be they a small enterprise or a couple of individuals, do not have such support. Independent developers may be driven away by larger 'competitors' that can afford patent litigation.

Last but not least, patent law does not have a fair-use defence provision comparable to the one embodied in copyright doctrine. Welfare losses resulting from the market power a patent holder exerts over his respective technology thus cannot be absorbed or mitigated. 59 When we talk about security, we have to admit that this lack of balance between private and public interests is highly questionable.

7. Conclusion and Recommendations

According to Richard Posner, 'legal rules, including the rules made by judges, are to be judged by their consequences' (Friedman D; 1998a, p. 55). In the case of intellectual property rights, that would call for some kind of assessment of the law with a focus towards security aspects. Modern IPR legislation should reflect the security needs of modern, networked societies. Until now, it does not. 60

7.1. The 'Source Code Privilege'

Perhaps, the proposal of a 'source code privilege' to be included in patent laws may provide a partial solution. 61 A 'source code privilege' for open source software was first proposed in an expertise prepared for the German Federal Ministry of Economics and Technology (Lutterbeck B et al.; 2000). The core proposal suggests (Recommendation PP-1): 'The use of the source codes of computer programs must be granted privileged status under patent law. The creation, offering, distribution, possession, or introduction of the source code of a computer program in its various forms must be exempted from patent protection (source code privilege)'.

7.2. Pros and Cons of the Source Code Privilege

Acknowledging the established system of intellectual property rights, the source code privilege does not overthrow the patent system. 62 Instead it is a sensitive proposal for its modification with the goal of recognizing the different characters of modern software development methods and their strengths and weaknesses. The needs of modern networked societies demand the exploration of all ways that may lead to more secure and more reliable software.

Some aspects of introducing a source code privilege into patent law will be discussed in the next section. Clearly, that discussion cannot be exhausting.

7.2.1. What is the Main Effect of the ‘Source Code Privilege’?

The source code privilege works as a fair use defence for those willing to open the code basis of their products, hence furthering quality and transparency. Even if patented technology is used -- intentionally or unintentionally -- developers and users of open source software are shielded from patent litigation so long as they are not using the code to make money. At the moment they are making use of a ‘software-related invention’ for more than development and distribution purposes they would have to pay royalty fees as usual. 63-64 As a consequence, the distribution of the costs of a patent search would be optimized insofar as only those who intend to commercially exploit the code would have to bear the costs.

7.2.2. Why Should We Create a Source Code Privilege Instead of Encouraging Open Source Software Developers to Patent Their Respective ‘Inventions'?

There are various reasons why open source software developers do not patent their ideas (‘inventions'). Beside the individual's moral decision not to claim exclusive rights, there are more objective reasons. The most important of which is the open source development process itself: It is open for all comers to look at how the code develops. But from a patent law point of view the result is the loss of novelty. Since only a few countries provide a novelty grace period within their patent laws, open source developers are barred from applying for patents. 65 They have to make a decision between developing their code according to the open source model and keeping their software-implemented inventions patentable. The general introduction of a novelty grace period into the patent laws of all countries could change this situation. 66 Owning patents, open source developers would be in a better situation for negotiating licences with other patentees. But even if a novelty grace period existed, the problems of inefficiencies of the patent application process as well as of the patent search process -- facing uncountable lines of code distributed in open archives all over the world -- remain. The combination of a ‘source code privilege' together with a ‘novelty grace period' looks most promising.

7.2.3. Can the Source Code Privilege Help to Overcome the Inefficiencies of the Patent System with Regard to Software?

The inefficiencies of the patent system with regard to software derive from subjective as well as objective circumstances. For example, the problem of insufficient funding of the patent office cannot be ‘healed' by a source code privilege. And the question of how to make a prior art search confronted with overwhelming masses of freely available source code won't be answered satisfactory. Software is usually developed incrementally, making it hard to draw a line, in patent terms, between what is new and what not. But the more source code was available openly, the better the state of the art would be documented. Patents lacking the required novelty could be more easily identified and invalidated by those interested in doing so. Thus, the welfare losses resulting from massive intellectual property protection for trivial ideas might decrease. The quality of the output of the patent office would increase accordingly.

7.2.4. But How to Address the Problem of Inefficient Licensing?

Inefficient management of property rights may lead to an ‘anticommons situation' leaving all parties in a less than optimal state (Heller; 1998). Inefficient licensing in the field of

software is the result of asymmetric market power distribution (big players vs. individual developers) as well as a ramification of the inefficient nature of the intellectual property system and the -- unavoidable -- inefficient work of patent offices under modern patent doctrine. 67 One industrial approach to solve the licensing problem is patent pools. However, patent pools have their downsides. They are problematic from a competition point of view and the costs of managing them are prohibitively high for individual developers or non-profit organizations. Complementing the source code privilege, Lutterbeck et al. (2000, p. 9) propose to implement structures allowing for the 'collective licensing' of patents (comparable to the collecting societies managing copyrights).

7.3. Concluding Words

Surely, many more questions arise from the arguments made in favour of the thesis discussed in this paper. Some of the arguments may need critical evaluation; others are underpinned by daily evidence. Undoubtedly, the proposed source code privilege presents nothing more than a pragmatic approach and by no means a systematic one. Still, it might provide a starting point for a discussion of what support modern information technology deserves from intellectual property laws.

We need to pay the required attention to the security needs of today's information infrastructures. Insofar as intellectual property rights are part of the problem, they should be 'bug-fixed'. 68 'Intellectual property law cannot escape evolutionary pressures that require adaptation and innovation as the price of survival.' (Reichman; 1994, p. 2558)

References

1. One critic, Lunney (2001), goes so far as to speak of 'bought-and-paid-for legislator[s]' (p. 898). See also Litman (2001), describing the history of the process that brought copyright enhancements about.

2. See, e.g., the comment of the Publishing Company Reed Elsevier (2000), Inc., in response to 65 FR 35673 (5/9/00): 'Certainly, the DMCA has helped to make computer networks safer but by no means risk-free places to distribute copyrighted works.' 65 FR 35673 asked for public comments on the intended agreement on the Uniform Computer Information Transaction Act (UCITA) in the U.S.

3. This behaviour was regularly criticised. See, e.g., Camp (2001, p. 1): 'Proposals to install mandated copy controls for the protection of distributed copies of high-value content have been called a step towards a secure network by some and a serious threat to security by others.' Samuelson and Scotchmer (2002, p. 1637) note: 'In particular, the [DMCA] may unduly impinge on fair and other non-infringing uses of digital content, on competition within the content industry, on competition in the market for technical measures, and on encryption and computer security research.' The Electronic Frontier Foundation, a US civil right activists organisation, recently has published a paper documenting the '[u]nintended [c]onsequences' of '[f]our [y]ears under the DMCA'. (EFF; 2003)

Assuming that public security indeed is a matter of politics for modern societies --and public health insurance, anti-terrorist laws, and other measures the like seem to indicate it-- the ignorance of IT security in modern IPR legislation is at least unreasonable. Lunney (2001, p. 914 f) is right, when he remarks: 'The price of turning our system of protecting creative works over to the private interests of copyright producers will prove high indeed'.

3. Quoted from a July 1st, 2002, e-mail to the Internet Security News mailing list; subject: [ISN] Microsoft's Digital Rights Management - A Little Deeper. (emphasis added; on file with the author)

5. MS security bulletin MS02-032. Microsoft has produced an updated version of the security bulletin (July 24, 2002) that is available online: <http://www.microsoft.com/technet/treeview/ default.asp? url=/technet/security/bulletin/MS02-032.asp> [02 Jan 2002]. The original ISN-message, however, still can be found online, e.g. at: <http://cert.uni-stuttgart.de/ archive/isn/2002/07/msg00007.html> [02 Jan 2002].

6. From a 'security-of-the-user' point of view one can consider this measure to be the preparation of the installation of a so-called Trojan horse, i.e. a program containing additionally malicious code performing unwanted functions.

7. See, e.g., Forno (2000), Schmidt (2001) or Schneier (2000, p. 159).

8. Important are the anti-circumvention provisions of the U.S. Digital Millennium Copyright Act (1998) and their European counterpart, the European Commission's Directive COM 2001/29/EC.

9. Most prominent is the UCITA in the U.S. For its expected implications on IT security see, e.g., IEEE USA (2000a) and IEEE USA (2000b).

10. Such figures always have to be taken cum grano salis. For example, Blakeley (2002, p. 2) complains that '[i]t's impossible to find out how many hours of computer downtime result from computer virus outbreaks in a year'.

11. There exists no comprehensive, all-purpose definition of security in the field of computer science. E.g., Viega and McGraw (2001, p. 14) note that '[s]ecurity means different things to different people. It may even mean different things to the same person, depending on the context'.

12. See, e.g., Viega and McGraw (2001, p. 14) or Blakeley (2002, p. 1).

13. A system consists of at least one piece of software running on at least one piece of hardware. Nevertheless, modern systems are usually networked and based on more than one piece of each, software as well as hardware; and people are involved, working with the system. For a more thorough discussion of what a system can be, see Anderson (2001a, p. 8 f).

14. For an in-depth discussion of software reliability see the classic work of Myers (1976).

15. Viega and McGraw (2001, p. 15) discuss if the relation between security and reliability is one of the first being a subset of the second.

16. According to Viega and McGraw (2001, p. 3): 'Software is at the root of all common computer security problems'.

17. For reasons of suitability neither step one nor step three will be discussed here in detail. A discussion of the shortcomings of step two, specification, and its ramifications will do for the purpose of this paper.

18. As Pipkin (2000, p. 70) puts it: 'Most security issues come from software with poorly designed security. Not that the software was poorly designed, more often security was never considered in the software design'. Similarly argue other authors, e.g., Viega and McGraw (2001, p. 15).

19. A short look into most of the standard books on software engineering reveals that designing software in a way to be secure is not a big issue. Most often, few pages of the respective book deal with matters of security. Those that do, almost always deal with the issue in a cursory way. Howard and LeBlance (2001) and Viega and Gary (2001) are two of the (very few) exceptions. Anderson (2001) goes further and explains how to build secure systems, i.e. a secure combination of hard- and software.

20. '[T] esting is largely a problem in economics'. (Myers; 1976, p. 176) What testing is considered necessary depends mostly on economical considerations rather than a technical ones. Today, about 40-50% of the development costs of a software product are caused by testing and debugging (Sommerville; 2001, p. 24). And testing and debugging costs grow faster than linear if more test conditions and environments are checked.

21. It is questionable if, in practice, it makes sense to speak of correctness in connection with testing. Hamlet (1995, p. 200) acknowledges the problem of having an oracle that would predict the correct outcome of a test. Without such an oracle it is hard to see how to decide if the result of a test can be judged the right way. '[I]n practice, there is no oracle but imperfect human judgement'. And an imperfect human judgement may in some cases accept a result as correct that in fact is not.

22. If we widen the focus for a moment, then, we will realise that systems (the combination of hardware, operating system, device drivers, static and dynamic libraries, . . .) instead of a single piece of software would need to be tested. Given that, the problem gets worse: '[M]odern systems have so many components and connections -- some of them not even known by the systems' designers, implementers, or users -- that insecurities always remain' (Schneier; 2000, p. xii).

23. 'In general, it is impractical, often impossible, to find all the errors in a program. This fundamental problem will ... have implications on the economics of testing ... ' (Myers;

1979, p. 8). See also Kaner (1997a).

24. Pipkin, for example, is right when he notes that, in order to test for security, '[t]he test procedures must be changed to focus on security issues such as testing for unexpected input, processing beyond endpoints, and the handling of invalid input' (2000, p. 74). The general rule for security testing is 'Expect the unexpected and test for the impossible' (ibid). Obviously, such a rule of paranoia is not a good basis to build a business on -- at least not for a profit-oriented business. Testing for the impossible while looking for the unexpected may well last forever…

25. That holds true in general. In particular cases, however, the absence of some specific types of errors may be shown by exhaustive enumeration. What cannot be demonstrated is the absence of conceptual errors and omissions of all kind.

26. Assuming bounded rationality here, would not considerably change the picture.

27. Perfect price discrimination could change the picture by adapting prices to more customers' preferences. However, today there is no perfect price discrimination in the software market. And it is doubtful if it ever will be, because 'price discrimination often arouses strong opposition from the public'. (Odlyzko; 2002)

28. This argument is in line with the assumptions underlying the 'homo economicus' model for economic behaviour.

29. Cf., e.g., Pipkin (2000, p. 75): '[T]he huge push to deploy software at breakneck speeds in Internet time leads to software systems being released and implemented with inadequate testing. Some software companies view problem reporting and bug fixing as software testing'. Viega and McGraw (2001, p. 17) take the same point of view: '[T]esting ... regularly ends up with no scheduled time and few resources. An all-too-common approach is to leave rigorous testing to users in the field (sometimes even paying users when they find bugs!)'.

30. The resulting economic losses and expenses explain the figures from section 1.1 above.

31. For a more complete understanding of information asymmetry in the software market, additional factors should be discussed, e.g., the institutional capacity of a big software company to collect information about customers' solvency and about liability regulations. The latter may even be influenced by donations to the election campaign funds of political candidates.

32. For an insightful discussion of the role network externalities (or network effects as they are also called) in the development of modern technology see Shapiro and Varian (1999).

33. 'By keeping its interface proprietary and by providing an exclusive set of applications, a platform owner has some hope of exploiting 'network effects' to become a de facto

standard in the market' (Samuelson and Scotchmer; 2002, p. 1617).

34. The every-day example is the exchange of text documents that come in proprietary formats. Who is able to resist this 'temptation'?

35. Says Lunney (2001, supra note 204, p. 877): 'Given that risk-taking is being subsidized, it should not be surprising to see the risk level increase'.

36. From a legal point of view, software is treated as a literary work (Raskind; 1998), as written speech. And in general you cannot be held liable for distribution of literary information that contains errors. That rule applies not only to books but to software too. There is no effective liability law to be applied in cases involving mass-market software and/or the online delivery of electronic goods (Kaner; 1996 and 1997b). Within the software market we can thus obey what Reichman (1994, p. 2453) predicted: '... the inefficiencies that copyright law tolerates in the specialized market for literary and artistic works are transferred to segments of the general products market'.

37. For a discussion see, e.g., Friedman (1998b) and Reichman (1994, p. 2520 ff).

38. For a broad discussion of the U.S. situation regarding the reverse engineering of computer programs and other technology see Samuelson and Scotchmer (2002).

39. That means the removal of errors is allowed without an appropriate licence when necessary to make the program running in the way it was intended by the vendor. Further removal of errors is not covered.

40. See, e.g., Article 6 of the European Copyright Directive (Directive 91/250/EC): 'The authorization of the right-holder shall not be required where reproduction of the code and translations of its form within the meaning of Article 4 (a) and (b) are indispensable to obtain the information necessary to achieve the inter-operability of an independently created computer program with other programs ... ' The intention of this clause is to enable competition among suppliers of compatible and complementary products, but not more (Bath; 2002, p. 143).

41. Regarding critical information infrastructures and anti-competitive behaviour, compulsory licences may be necessary in order to establish a second source of supply for a certain technology. The WTO TRIPs agreement outlines rules according to which member states may implement laws that, in very limited circumstances, provide for 'use without authorization of the right holder' (Art. 31). In fact, most national patent laws already provide for compulsory licences. In practice, however, courts and legislators are reluctant to restrict private property rights and therefore compulsory licences are rarely (almost never) used. For example, in the 'patent history' of the post-war Germany, it occurred only one time that a compulsory licence was granted. The compulsory licence was revoked sometimes later. (Schwendy; 1999, p. 403)

42. The problem of a comprehensive prior art search has yet to be resolved for software technology. There are huge amounts of commercial and non-commercial software

floating around that are not examined during the patent application process. At the same time there is practically no way to receive reliable assurance that a patent does not protect specific software technology. Software can simply fall into too many categories of patents. At most a patent search barely increases the probability that a certain technology is not the subject matter of a patent. The problem discussed here gets much worse when one takes into account the side-effects of the doctrine of equivalents., as it exists in some countries, e.g., in the U.S., the country with the largest software industry.

The doctrine of equivalents (in Germany: 'Equivalenztheorie') is a peculiarity of the respective national patent system whereby one can infringe a patent in a third way (beside actual or contributory infringement) by deploying technology not identical but equivalent to the one described in the patent claims. In short, the 'doctrine of equivalents' widens the scope of validity of the patent beyond what is literally written into the patent claims. Most recently, the U.S. Supreme Court in Festo Corp. v. Shoketsu Kogyo Kabushiki Co., Ltd., 122 S. Ct. 1831 (2002), overturning an en banc decision of the Federal Circuit of Appeals (234 F.3d 558, 56 USPQ2d 1865 (Fed. Cir. 2000)), supported a flexible interpretation of the doctrine of equivalents. A flexible interpretation results in a stronger position of the patent holder -- and higher uncertainty for all others. Both decisions are available online: <http://caselaw.lp. findlaw.com/cgi-bin/getcase.pl?court=US&navby=case&vol=000 &invol=00-1543> [02 Jan 2003]. For a short and instructive discussion of the issue see (Mota S A; 2002).

43. Landwehr (2002, p. 2) argues: '[T]he lack of specific information about the ability of specific components and system architectures to preserve information availability, integrity, and confidentiality in the face of failures and attacks, and the difficulty of developing this information quickly, is a strong factor in the current state of computer system security, or the lack thereof ..'.

44. The RSA encryption technology, invented by R.L. Rivest, A. Shamir, and L. Adleman, gives an example of how the market penetration of a superior technology -- strong public key encryption in the case of RSA -- suffers from patent protection and according restrictive licensing policies. RSA was patented only in the United States (U.S. Patent 4,405,829, 20 Sep 1983), but not in other countries. In the result, RSA became a (royalty-free) de-facto-standard in large parts of the world but was less successful in the U.S. (Schneier; 1996, p. 540 f). And Atkinson and Klinker (1999, p. 221) note: '[T]he vendor and user communities generally prefer to avoid patented technologies (e.g. RSA) due to the perceived higher costs'. The expiration of the patent in 2000 was welcomed especially, but not only, by open source protagonists: 'The big news in security is the expiration of the RSA patent, which, along with the U.S. government's relaxation of export controls, means we can use open-source security tools such as OpenSSL everywhere. No more paying tribute to the RSA bandits to use their patent. That's great news, and we expect the Linux distributions to start integrating strong crypto everywhere' (Marti; 2000).

On the contrary, Guntersdorfer and Kay (2002) argue '[h]ow [s]oftware [p]atents [c]an [s] upport COTS [c]omponent [b]usiness', and improve existing software technology. Their main arguments are (1) the necessary protection of 'truly novel software ideas from unfair

exploitation'; (2) the effect of patents to release 'the knowledge to the community', thus working 'against secrecy'; and (3) the necessity to licence technology (instead of re-implementing it) resulting from patent protection that would 'support scientific progress, which, in the case of software, includes software reuse' (p. 79). They argue further that '[p]atent protection buys inventive programmers lead time that lets them produce higher-quality products using better but more time-consuming software engineering methods'. Neither do the authors discuss the downsides of the resulting market distortion nor the implications for open source developers. The question is not asked, why a patentee would prefer to make investments in product quality (in the absence of effective liability and competition) instead of reaping the additional profits resulting from its state-protected monopoly position.

45. See, e.g. Finjan Software, Inc. (San Jose, CA), U.S. Pat. 6,167,520 (26 Dec 2000): System and method for protecting a client during runtime from hostile downloadables; McAfee.com Corporation (Santa Clara, CA), U.S. Pat. 6,266,774 (24 Jul 2001): Method and system for securing, managing or optimizing a personal computer.

46. Small and medium enterprises as well as individual and institutional developers regularly won't be able to afford negotiations. The situation looks better for large companies with strong patent portfolios enabling them to negotiate cross-licence agreements instead of single licences.

47. For example, Fisk (2002, p. 2) stresses the importance of human expertise: 'Given the current state of formal software testing and proving, improving or maintaining the quality of a software system is primarily a manual process affected by the quality and quantity of expertise applied to it'.

48. For those not familiar with the open source software development model, DiBona et al. (1999) and Raymond (1999) contain a number of introductory essays.

49. Unrestricted is not entirely true. Quite often there are restrictions on the appropriation of the software covered by an open source licence. The intention is to keep the source code open to all comers, a vital condition for the sustainability of the development process.

50. Compare this to the proprietary model where, in general, only binary code is delivered to the users. For almost all users binary code is all but transparent.

51 This characteristic may prove especially helpful in making computer networks more secure. As Schneier (2002, p. 3) notes: 'Internet security is a commons'. A strategy for software development intended to create and sustain the digital commons may be the right choice.

52. Differing from the material world, the goods to be shared are characterized by non-rivalry (everything is copied) and non-exhaustion (digital copies are lossless). 'Virtual overgrazing' (Hardin 1968) is simply impossible. Every share increases the digital commons. Compare this to the material world where every occupation of a part decreases

the (size and value of the) commons.

53. The viability under changing conditions and the growth of the digital commons depends on continuous contributions by those who are sharing it. Licences that demand the sharing of enhancements guarantee further growth. From an economic point of view every new act of sharing constitutes a new Pareto-optimal situation: The one who makes the contribution is not worse-off due to the digital nature of the object of his donation. He simply makes a copy. When at least one other user profits from the new share, i.e. uses the copy, society, as a whole, is better off.

54. As Hamlet (1995, p. 195) notes, process improvement and product improvement are two different things. Finding more failures through testing is a matter of process improvement, while removing the errors has impact on the final product.

55. That means both availability of source code and appropriate licence conditions allowing for the modification of the code.

56. For Landwehr (2002, p. 2) open source has the advantage to allow for faster spreading of information: 'Security information about proprietary software often takes longer to develop because only the proprietor has unrestricted access to the code and so the decision of whether to apply resources to security analysis of it is constrained. Opening source permits anyone who cares to apply resources to this task to do so'.

57. If one follows the arguments of Fisk (2002) who proposes to hold system owners liable if their systems became part of an attack to other systems, it makes sense to give opportunities to the system owners to protect their systems, what exactly open source code does. Clearly, there is lack of empirical work investigating if the acceleration in fixing bugs by opening the source code really happens as presumed theoretically in this paper (and elsewhere). But, for example, the study of Ritchey (2001) seems to point in that direction.

58. See, e.g., Pfitzmann et al. (2000) and Schneier (2000). Others argue that neither the closed source nor the open source model offers significant advantages (Viega J and McGraw G; 2001). However, they don't discuss the economics of software development and their consequences.

59. Samuelson and Scotchmer (2002, p. 1646) underline the importance of competition in order to reduce welfare losses: '[T]he vulnerability of unpatented products to reverse engineering limits market power in a competitively healthy way'.

60. The 'state security' clauses in patent laws are hardly useful in the context of this discussion. The do refer to the state where network security refers to networks often transcending national boundaries and legislation.

61. Presented in the short expertise on 'Security in Information Technology and Patent Protection for Software Products: A Contradiction?', Commissioned by the Federal Ministry of Economics and Technology (Lutterbeck et al.; 2000).

62. '... if we did not have a patent system, it would be irresponsible, on the basis of our present knowledge of its economic consequences, to recommend instituting one. But since we have had a patent system for a long time, it would be irresponsible, on the basis of our present knowledge, to recommend abolishing it'. Fritz Machlup (1958) quoted with (Kitch; 1998, p. 14).

63. I use the terminology preferred by the European Commission.

64. Other exemptions of the patent law for personal use, research, etc. would apply as well.

65. E.g., the U.S. patent law has a novelty grace period.

66. I made this proposal in Gehring (2000); Lutterbeck et al. (2000) supported it.

67. See section before.

68. Some observers see the ramifications of modern IPR legislation for public interests as 'bug[s] in the legal code', as Damien Cave (2000) put it.

Books, Magazine and Journal Articles

Akerlof G A (1970) 'The Markets for "Lemons" -- Quality Uncertainty and the Market Mechanism', Quarterly Journal of Economics, vol. 84, no. 3, pp. 488-500.

Anderson R J (2001a) Security Engineering. A Guide to Building Dependable Distributed Systems (New York: John Wiley & Sons).

Anderson R J (2001b) 'Why Information Security is Hard -- An Economic Perspective', <http://www.cl.cam.ac.uk/ftp/users/rja14/econ.pdf> [28 Aug 2001].

Atkinson R J and Klinker J Eric (1999) 'Progress in Internet Security' in Zelkowitz, M V (ed) Advances in Computers, vol. 48, pp. 219-255.

Blakley B (2002) 'The Measure of Information Security is Dollars', Position paper, First Workshop on Economics and Security, May 2002, Berkeley, <http://www.sims.berkeley.edu/resources/affiliates/workshops/econsecurity/econws/54.pdf> [11 Jan 200].

Bath U (2002) 'Access to Information v. Intellectual Property Rights', European Intellectual Property Review, vol. 24, issue 3, March 2002, pp. 138-146.

Camp L J (2002) 'Marketplace Incentives to Prevent Piracy: An Incentive for Security?', Position paper, First Workshop on Economics and Security, May 2002, Berkeley,

<http://www.sims.berkeley.edu/resources/affiliates/workshops/econsecurity/econws/29.txt> [11 Jan 2002].

Cave D (2002) 'A bug in the legal code?', Salon Magazine, Sep 13, 2000, <http://www.salon.com/tech/feature/2000/09/13/touretzky/> [14 Sep 2002].

'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', Official Journal L 167, 22/06/2001 P. 0010 - 0019, <http://europa.eu.int/eur-lex/en/lif/dat/2001/en_301L0029.html> [22 Feb 2002].

DiBona C, Ockman S and Stone M (1999) 'Open Sources. Voices from the Open Source Revolution', (Sebastopol: O'Reilly).

EFF (2003) 'Unintended Consequences: Four Years under the DMCA', <http://www.eff.org/IP/DMCA/20030102_dmca_unintended_consequences.html> [12 Jan 2003].

Fisk M (2002) 'Causes & Remedies for Social Acceptance of Network Insecurity', Position paper, First Workshop on Economics and Security, May 2002, Berkeley, <http://www.sims.berkeley.edu/resources/affiliates/workshops/econsecurity/econws/35.pdf> [11 Jan 2002].

Floyd C and Züllighoven H (1997) 'Eigenschaften von Software' in Rechenberg, P and Pomberger, G. (eds) Informatik-Handbuch, p. 644 (München: Carl Hanser Verlag).

Forno R (2000) 'Microsoft: A Proven Danger to National Security: Conclusions of Reality', Essay no. 2000-4, May 2000, <http://www.info-sec.com/internet/00/MSFOR_natsec.pdf> [17 Oct 2001].

Friedman D D (1998a) 'Posner, Richard Allen' in Newman, P (ed) The New Palgrave Dictionary of Economics and The Law, vol. 3, pp. 55-62 (London: MacMillan Reference).

Friedman D D (1998b) 'trade secret' in Newman, P (ed) The New Palgrave Dictionary of Economics and The Law, vol. 3, pp. 604-607 (London: MacMillan Reference).

Gehring R A (2000) 'Berliner Ansatz zu "Open Software Patents". Ein Ausweg aus dem "Digital Dilemma"?', Position paper, Conference on Economic Aspects of Software Patents, German Federal Ministry of Commerce, May 18, 2000, Berlin.

Guntersdorfer M S and Kay D G (2002) 'How Software Patents Can Support COTS Component Business', IEEE Software, vol. 19, No. 3, pp. 78-83.

Hamlet D (1995) 'Software Quality, Software Process, and Software Testing' in Zelkowitz, M V (ed) Advances in Computers, vol. 41, pp. 191-229.

Hardin R (1968) 'The Tragedy of the Commons', 162 Science 1243. [Heller M A; 1998] Heller, M A (1998) 'The Tragedy of the Anticommons: Property in the Transition from Marx to Markets, 111 Harvard Law Review 623.

Heller M A and Eisenberg R S (1998) 'Can Patents Deter Innovation? The Anticommons in Biomedical Research', 280 Science Magazine 698.

Howard M and LeBlance D (2001), Writing Secure Code, Microsoft Press, 2001

IEEE USA (2000a) 'Opposing Adoption of the Uniform Computer Information Transactions Act (UCITA) By the States, Approved By the IEEE-USA Board of Directors', Feb. 2000, <http://www.ieeeusa.org/forum/positions/ucita.html> [28 Aug 2001].

IEEE USA (2000b) 'Recommended Amendments to Virginia Uniform Computer Information Transactions Act, (Title 59.1, Chap. 43, $ 59.1-501.1 et. seq.)', Oct 2000, <http://www.ieeeusa.org/forum/POLICY/2000/00oct17.html> [28 Aug 2001].

Kaner C (1996) 'Liability for Defective Content', Software QA, vol. 3, no. 3, p. 56, <http://www.badsoftware.com/badcont.htm> [28 Aug 2001].

Kaner C (1997a) 'The Impossibility of Complete Testing', Software QA, vol. 4, no. 4, p. 28, <http://www.kaner.com/articles.html> [28 Aug 2001].

Kaner C (1997b) 'Software Liability', <http://www.kaner.com/articles.html> [28 Aug 2001].

Kaner C (1998) 'The Problem of Reverse Engineering', Software QA, vol. 5, no. 5, <http://www.kaner.com/articles.html [28 Aug 2001].

Kitch E W (1998) 'Patents' in Newman, P (ed) The New Palgrave Dictionary of Economics and The Law, vol. 3, p. 13-17, (London: Macmillan Reference).

Landwehr C E (2002) 'Improving Information Flow in the Information Security Market', Position paper, First Workshop on Economics and Security, May 2002, Berkeley, <http://www.sims.berkeley.edu/resources/affiliates/workshops/econsecurity/econws/11.doc> [11 Jan 2002].

Levinson M (2001) 'Let's Stop Wasting $78 Billion a Year', CIO Magazine, October 15, 2001, <http://www.cio.com/archive/101501/wasting_content.html?printversion=yes> [16 Oct 2001].

Litman J (2001) Digital Copyright (Amherst: Prometheus Books).
Lunney G S Jr (2001) 'The Death of Copyright: Digital Technology, Private Copying, and the Digital Millennium Copyright Act', 87 Virginia Law Review 813.

Lutterbeck B, Horns A H and Gehring R A (2000) 'Sicherheit in der

Informationstechnologie und Patentschutz für Softwareprodukte - ein Widerspruch?', Short Expertise Commissioned by the Federal Ministry of Economics and Technology, Berlin, December 2000, <http://www.sicherheit-im-internet.de/download/Kurzgutachten-Software-patente.pdf> [28 Aug 2001].

Marti D (2000) 'Focus: Security', p. 91, Linux Journal 10/2000.

McAfee (2001) 'McAfee Lösungen', <http://www.mcafeeb2b.com/international/germany/products/mcafee-solutions.asp> [23.08.2001].

Mota S A (2002) 'The Doctrine of Equivalents and Prosecution History Estoppel: The Supreme Court Supports Flexibility Over Certainty in Patent Cases in Festo v. SMC', Richmond Journal of Law and Technology, vol. IX, issue 1, Fall 2002, preprint, <http://law.richmond. edu/jolt/prepub/v9i1/article2.pdf> [02 Jan 2003].

Myers G J (1976) Software Reliability. Principles and Practices (New York: John Wiley & Sons).

Myers G J (1979) The Art of Software Testing (New York: John Wiley & Sons).

Odlyzko, A (2002) 'Privacy, Economics, and Price Discrimination on the Internet', Position paper, First Workshop on Economics and Security, May 2002, Berkeley, <http://www.sims.berkeley.edu/resources/affiliates/workshops/econsecurity/econws/52.txt> [11 Jan 2002].

Pfitzmann A, Köhntopp K and Köhntopp M (2000) 'Sicherheit durch Open Source? Chancen und Grenzen', pp. 508-513 in Datenschutz und Datensicherheit 9/2000.

Pipkin D L (2000) Information Security (Upper Saddle River: Prentice Hall PTR).

Raskind L J (1998) 'Copyright' in Newman, P (ed) The New Palgrave Dictionary of Economics and The Law, vol. 1, p. 478-483 (London: Macmillan Reference).

Raymond E S (1999) The Cathedral & The Bazaar. Musings on Linux and Open Source by an Accidental Revolutionary (Sebastopol: O'Reilly).

Reed Elsevier, Inc. (2000) 'Response to 65 FR 35673', <http://www.loc.gov/copyright/reports/studies/dmca/reply/Reply005.pdf> [29 Aug 2001].

Reichman J H (1994) 'Legal Hybrids Between the Patent and Copyright Paradigms', 94 Columbia Law Review 2432.

Ritchey R W (2001) 'Open Source Vs. Close Source Software. An Experiment To Determine Which is More Secure', Study Paper, Dec 2001, <http://www.isse.gmu.edu/faculty/ofut/ classes/763/studpapers/Ritchey.pdf> [22 Dec 2002].

Rosenoer J (1997) Cyber law. The Law of the Internet (New York: Springer-Verlag).

Samuelson P and Scotchmer S (2002) 'The Law and Economics of Reverse Engineering', 111 Yale Law Journal 1575.

Schechter S (2002) 'Quantitatively Differentiating System Security', Position paper, First Workshop on Economics and Security, May 2002, Berkeley, <http://www.sims.berkeley.edu/resources/affiliates/workshops/econsecurity/econws/31.pdf> [10 Jan 2002].

Schmidt J (2001) 'Sicherheitsrisiko Microsoft. Die Kehrseite des Windows-Komforts', pp. 140-142 in c't Magazin 21/2001.

Schneier B (1996) Angewandte Kryptographie. Protokolle, Algorithmen und Sourcecode in C (Bonn: Addison-Wesley).

Schneier B (2000) Secrets and Lies. Digital security in a networked world (New York: John Wiley & Sons).

Schneier B (2002) 'Computer Security: It's the Economics, Stupid', Position paper, First Workshop on Economics and Security, Berkeley, May 2002.

Schwendy K (1999) $ 24 pp. 400-413 in Keukenschrijver A, Schwendy K and Baumgärtner T (ed) Busse: Patentgesetz. Kommentar, 5th ed (Berlin, New York: Walter de Gruyter).

Shapiro C and Varian H R (1999) Information rules. A strategic guide to the network economy (Boston: Harvard Business School Press).

Sommerville I (2001) Software Engineering, 6th Edition (German translation) (Pearson Studium).

Thibodeau P (2002) 'Study: Buggy software costs users, vendors nearly $60B annually' in Computer World, June 25, 2002, <http://www.computerworld.com/managementtopics/management/itspending/story/0,10801,72245,00.html> [26 Jun 2002].

Viega J and McGraw G (2001) Building Secure Software. How to Avoid Security Problems the Right Way (Boston: Addison-Wesley).