Legal Issues Relating to Free and Open Source Software

Editors

PROFESSOR BRIAN FITZGERALD
Head of the School of Law
Queensland University of Technology, Australia

GRAHAM BASSETT
Barrister, Bank of New South Wales Chambers, Brisbane, Australia

Essays in Technology Policy and Law Volume 1

Queensland University of Technology
School of Law

ISBN 0-9751394-0-1

Contents

Preface

This publication was given impetus by my spending a sabbatical leave in the heart of Silicon Valley at Santa Clara University Law School and Red Hat opening a major office in my hometown of Brisbane Australia.

In Silicon Valley I was able to draw upon a wealth of free and open source practitioners – a number of them like Larry Rosen and Bill Lard are Santa Clara alumni – to join with me in running a public seminar on Legal and Business Issues for Free and Open Source Software on 7 June 2001 much of which is embodied in Chapter 3 "Live from Silicon Valley".

On returning to Australia and learning that a number of lawyers in Australia including Martin McEniery and Anne Fitzgerald (my sister) were acquaintances of Mark Webbink, legal counsel for Red Hat, it was decided that we would run a conference at my Law School at the Queensland University of Technology in Brisbane Australia.

All the while the genius of the free software movement was spreading throughout the world like wild fire most obviously in the form of the Linux operating system. The powerful insight that Richard Stallman and his associates at the Free Software Foundation had discovered was that if you want to structure open access to knowledge you must leverage off or use as a platform your intellectual property rights. The genius of Stallman was in understanding and implementing the ethic that if you want to create a community of information or creative commons you need to be able to control the way the information is used once it leaves your hands. The regulation of this downstream activity was achieved by claiming an intellectual property right (copyright in the code) at the source and then structuring its downstream usage through a licence (GPL). This was not a simple "giving away" of information but rather a strategic mechanism for ensuring the information stayed "free" as in speech. It is on this foundation that we now see initiatives like the Creative Commons expanding that idea from open source code to open content.

On 3 July 2002 I convened the Legal Issues Relating to Free and Open Source Software Conference which was opened by The Honourable Paul Lucas MP, Minister for Innovation and Information Economy for the State of Queensland Australia. The conference examined legal and business issues facing the development and implementation of free and open source software. The presenters were lawyers, academics and software developers expert in the area drawn from Australia and the USA.

The Program for the conference was as follows:

- "Welcome Address" — The Honourable Paul Lucas MP (Minister for Innovation and Information Economy)
- "An Overview of Free and Open Source Software Licences" — Professor Brian Fitzgerald and Graham Bassett
- "Legal Issues Arising from the Use of Free and Open Source Software in Business" — Mark Webbink
- "Live from Silicon Valley" — David Schellhase, Larry Rosen and Bill Lard
- "Security and Free and Open Source Software: The SE Linux Experience" — Professor Bill Caelli
- "The Developer's Perspective" — Andrew Tridgell, Paul Gampe and Rhys Weatherley
- "Advising on Free and Open Source Software: An Australian Perspective" — Peter James and Martin McEniery

This publication embodies much of what was presented at the conference, which was and still is contemporary and challenging. The issues considered not only reflect on the past and present but also the future landscape. It is interesting to see that Australia in 2003 is one of a few countries in the world that has Bills before its legislative bodies to mandate the consideration of free and open source software by government departments and agencies.

Note that Chapters 2, 4 and 5 represent fully prepared academic articles while Chapters 1, 3 and 6 are revised versions of the conference transcript. In my view they work extremely well together. Marks Webbink's practical comments work with Peter James focus on the operation of the GNU GPL in Australia which is followed by an excellent analysis by Bill Caelli of security and open source issues which in turn is followed by the very real comments and concerns of leading software developers.

The issues featured in this volume are directly relevant to legislators and government officers, academics and practitioners/professionals in the areas of law, business and information technology, as well as to developers of both proprietary and free and open source software.

I hope you enjoy reading this volume as much as I have enjoyed learning about the free and open source software model and the promise it provides not only for software development but for the dissemination of knowledge more generally.

Professor Brian Fitzgerald
Head of Law School QUT
Brisbane
20 September 2003

Acknowledgements

The QUT Conference and this publication would not have been possible without the generous support of Minister Paul Lucas and the Queensland Government Department of Innovation and Information Economy, QUT Law Faculty, Santa Clara Law School, Red Hat and the individual presenters and participants at the conference.

The enthusiasm and assistance of my co-convenor and co-presenter Graham Bassett was unwavering and exceptional as was the operational support of my assistant Suzanne Lewis who worked hard to ensure these projects were successfully completed.

The QUT Conference was invigorated by the brilliant minds of technology gurus like Bill Caelli and Andrew Tridgell and the pragmatic legal advice of Mark Webbink from Red Hat, who gave generously of their time as did all of the presenters: Minister Paul Lucas, Peter James, Martin McEniery, Larry Rosen, Bill Lard, David Schellhase, Paul Gampe and Rhys Weatherley.

Bill Lard ensured we had a link into one of the Sun campuses in Silicon Valley and allowed us a memorable and near perfect link to Lard, Rosen and Schellhase: Live from Silicon Valley. Dating back to the Santa Clara Conferences I owe these three very bright lawyers a huge vote of thanks. QUT AV technicians in particular Dennis Clark (Technical Development Coordinator) and Ross Hutton (Audio Visual Technology Assistant) made sure the link to Silicon Valley worked.

Lawyers Adrian McCullagh and Anne Fitzgerald, QUT Vice Chancellor Professor Peter Coaldrake and QUT Deputy Vice Chancellor Tom Cochrane all supported the conference through their attendance and involvement in sessions. Carl Middlehurst gave me some good start-up advice on contacts in Silicon Valley, while Nic Suzor, Cheranne Bartlett and Sally Hawkins provided first-rate research assistance.

Thank you.

Professor Brian Fitzgerald
Head of Law School QUT
Brisbane
20 September 2003

Foreword

THE HONOURABLE PAUL LUCAS
Queensland Minister for Innovation and Information Economy

I am pleased to be here today to welcome you to this important conference on "Legal Issues Relating to Free and Open Source Software". As you would be aware, Open Source Software has been the subject of international debate since a young hacker named Linus Torvalds developed Linux as a hobby in the early 1990s. Having been a lawyer before I became a Minister I recognise, on the one hand, Open Source Software encourages innovation and entrepreneurship but, on the other hand, it raises legal issues which must be addressed in a way that helps to promote commercial development of the ICT industry.

Today is an opportunity to revisit this debate and look at the legal issues associated with Open Source Software in the 21st century. The Beattie Government Smart State strategy is about seizing the challenges that come with the information age.[1] We recognise that ICT and associated software is having a huge impact across industry and our daily lives. ICT impacts on how we do business and work, how we are educated and how we are entertained. We only need to look at our local film industry to see how the information age is driving development across industry sectors while also changing how we are entertained. In 1999 the Queensland Government introduced the Queensland Communication and Information Strategic Plan 1999-2004 as a blueprint for driving government activity and ICT.[2] The second annual report[3] and the plan's progress shows that Queensland Government agencies are making significant progress in delivering on the strategies for the key areas of ICT skills, ICT industry development, e-commerce and telecommunications infrastructure. The Queensland Government sponsorship of today's forum further demonstrates our commitment to developing and pursuing all aspects of the ICT industry. I think it is important to mention that Queensland houses the largest e-security research community in the southern hemisphere and the largest one in the world besides that of the United States. The products we are

[1] The aim of the Smart State Initiative is to "develop Queensland as an Asia-Pacific hub for the new industries of the 21st Century – industries such as biotechnology, information technology, nanotechnology and communication technology". See: The Smart State, [http://www.thepremier.qld.gov.au/smartstate/index.htm].

[2] This plan "sets out the Queensland Government's blueprint, in partnership with business, industry and the wider community, to achieve its vision of *Queenslanders participating in the information age*". See: [http://www.iie.qld.gov.au/infoecon/stratplan.asp]

[3] See: Queensland Communication and Information Strategic Plan Progress Report 2001, [http://www.iie.qld.gov.au/publications/comminfo/default.asp#prog2001].

generating here in Queensland are regarded as the best in the world and this industry is growing from strength to strength.

Since 11 September 2001 many companies are developing ICT risk management strategies. The Queensland Government is moving to reinforce its already strong ICT security. Without going into detail, people should be reassured that a considerable amount of effort is expended to ensure Queensland has IT systems that are as secure as possible. According to the annual PWC technology forecast released in May, one of the challenges facing Australian companies over the next 2 years will be evaluating potential new software components and business applications.[4] The report revealed that CEOs will have to make some hard decisions about moving from current solutions to next generation systems and that there is a global trend towards increasing use of Open Source Software as an alternative to commercially licenced products.

At this point it is unclear whether this move is signalling a culture shift where companies recognise the role of innovation as well as cost efficiency in commercial competitiveness. Regardless, this is good news for the Open Source movement that has been prevalent in the technical culture for years but is only now breaking out into the commercial world. Allowing source code to be distributed freely and developed, used, copied or modified is certainly conducive to innovation and entrepreneurship that is vital to global competitiveness in the 21st century.

Open Source software provides a cost effective solution to start-up companies and provides a forum for testing ideas before they go to market. It allows for quick and innovative desktop solutions where the consequences are minimal if anything goes wrong or if a piece of software ceases to exist or be upgraded. However, history shows that freeware does not remain free for long once support is needed and there are claims that Open Source Software threatens security and of course legal rights. Combining free software with commercial software could violate a company's intellectual property rights and begs the question of where liability rests if something goes wrong, especially where third parties are involved.

I will not attempt to go into the ins and outs of these issues as that is what you will be doing today and you will have some international experts who are here and who will speak from first hand experiences in the field. I would like to mention that the premise underlining Open Source Software is similar to the position that the Queensland Government has taken in developing its Information Standard No. 33 – Information Access and Pricing.[5] This standard promotes a wide dissemination of

[4] PriceWaterhouseCoopers Annual Technology Forecast, "Aust CIOs becoming business clairvoyants", ZDNet Australia, 27 May 2002, www.zdnet.com.au

[5] "Government information must be made accessible, directly or indirectly, to citizens of Queensland and those doing business in Queensland at no more than the cost of provision and in a manner which provides reasonable access to the community unless statutory requirements vary the access and pricing arrangements. Certain types of information will be

Queensland public sector information as a key to realising the Smart State vision with an information economy. The rationale is to free up Government information held by agencies and to provide it at minimal cost to Queenslanders and people doing business in Queensland. This is consistent with a notion of free software contributing with the exchange of ideas and the public good. But, at the same time, the Government seeks to apply best practice principles to all of its operations which at this point commands priority. The situation demonstrates how the issues associated with Open Source Software are often double-edged swords. At best we should strive for a model where Open Source and proprietary software can co-exist for the benefit of all stakeholders within the ICT industry and across the many sectors that are impacted. It is vital that a legal framework is developed to support the model. While Open Source software provides a cost efficient and innovative solution for many companies it also raises legal and security issues that are yet to be fully addressed. The challenge lies in striking a balance between the needs of a range of stakeholders and developing an appropriate legal framework.

I look forward to hearing about the outcomes of the open and frank discussions that you will be participating in here today.

---

required to be freely available." See: Information Standard No. 33 Information Access and Pricing, [http://www.iie.qld.gov.au/comminfo/downloads/is33.pdf].

Chapter 1

Licensing and Open Source*

MARK WEBBINK
Senior Vice President and General Counsel, Red Hat, Inc., USA

It is exciting to be here also because of a number of reasons that are unique to Australia. First of all is the strong business climate that you are all enjoying at this time. I read recently that the Australian economy is one of the strongest economies in the world right now. It is not the USA economy where it seems that on a daily basis we are learning of one more management group of a major company that has undermined their own creditability and undermined the creditability of the US business by some of their financial reporting. I will have to say that I am proud of my own company for the diligence in which it records its financial transactions and in the openness of that reporting.

It's also exciting to be here because of the open-minded embrace of Open Source in Australia by the Government, by industry and by academia. This is something that you do not find right now in the United States, and quite frankly, there is a good reason for it. It's because there is one company that owns about 96% of the desktops in the United States that would rather not see Open Source succeed, and they are giving it their best shot to make sure that that does not happen. Finally, it's exciting to be here because you all provide evidence about what is the best about Australia. Among those characteristics that I observe are the strong character of the Australian people and the competitive spirit often times exhibited amazingly in athletic events. For the relatively small population of Australia, the athletes perform at incredible levels. I attribute that to the competitive spirit of the people here. And a final factor is your leadership in applying innovation, not just innovation, but applying innovation.

When I first joined Red Hat in May 2000 I am afraid I was a little bit of a die-hard in terms of what my desktop was going to be because I was working on legal agreements with other companies. I said: 'I have got to have Microsoft Word'. I could not deal with the documents back then without it. There were several desktop applications being used at Red Hat at the time. One loaded on the machine I had was called Applix which was just horrible. I said to them: 'I can either be your general counsel and get work done or I can spend all my time trying to find documents and trying to update them and then ship them out to people who will

---

* This is a revised version of the transcript from the Legal Issues for Free and Open Source Software Conference held at QUT in Brisbane Australia on 3 July 2002.

never be able to read them because you cannot ship them out in a document format they can read'. So for the next 15 months I was one of the few folks at our corporate headquarters that actually ran a Windows desktop. By last fall I just had about all I could stand and, thank goodness, Sun released a better version of Star Office, the 6.0 beta. I said, 'In for a penny, in for a pound, I am making a change now'. And granted Star Office is not Open Source in the 6.0 Release, but at least it runs on Linux and I can use it. What I found is that it works just fine for my purposes.

At Red Hat we have our internal email called Memo List. Memo List can only be experienced first hand. Your first experience with it is generally your worst. Hackers at Red Hat are very free with their advice. While they will often conclude a message with the letters *ianal* – 'I am not a lawyer' – that does not mean that they do not still have strong opinions about what the law ought to be, not necessarily what it is. Debates rage on Memo List. When you first get to Red Hat you think this is a normal corporation and you post a new policy statement. Then you find 500 people attacking your policy and telling you what is wrong with it. But in time you learn to channel that energy. I always make sure that in my notes to folks I end with *iaal* – 'I am a lawyer' and, if anything, I should have an attribution there IANAH – 'I am not a hacker'. But with that background and with one final caveat that my remarks today are based on US law (I have not been admitted to practise here in Australia, although I was scanning the classifieds this morning to look at all the positions at the various law schools and thinking how tempting they were). Some of what I will say can only be appreciated in the context of US law and as only a US lawyer can appreciate it. I will also provide the caveat that this is not specifically legal advice; this is simply general legal commentary. If you want specific legal advice see Brian Fitzgerald or Martin McEniery or some of the others who are here.

WHAT IS COPYRIGHT?

I am going to walk through some background on copyright. For those of you who practise law, you will think this is highly trivial but I want to set some context here. Because I will often have people write to me, and they will say "so and so is violating our copyright" and, in fact, what they are violating is our trademark rights or there is a patent issue.

So let us just walk through some very fundamental things about intellectual property. Copyright, patents and trademarks have some very fundamental distinctions among them. Patents protect inventions in the form of innovative or improved products or processes. Patents do not protect ideas, they protect the application of those ideas. Trademarks are a source identifier. Trademarks may be applied to goods and processes, whether or not patented, to identify those goods or services as coming from a particular party. Trademarks may also be used within copyrighted materials to form advertisements for promotions.

By contrast, copyright protects the original expression of an idea. Software is protectable under all three of these regimes. The source code of the software is copyrighted at the time it is written. It may be patentable if some or all of the software embodies innovative processes or algorithms. As it is promoted for sale it may have a trademark associated with it to identify the source of that software.

How does copyright arise? Under US law copyright arises immediately upon the work being fixed in a tangible medium of expression – that includes being recorded in an electronic form or on a computer or disk even though it cannot be reproduced or read without that computer or disk.

What rights does the owner of the copyright work have? Pay close attention to these as we walk through them. The owner of a copyrighted work has the exclusive right to:

- reproduce the work,
- distribute the work,
- create derivative works,
- perform the work, or
- display the work.

As we will see, these concepts are important to Open Source and free software.

WHAT IS OPEN SOURCE SOFTWARE?

We talked about this a little bit. But here is the definition provided by the Open Source Initiative. It contains the following rights and obligations:

- No royalty or other fee imposed upon redistribution,
- Availability of the source code,
- Right to create modifications and derivative works,
- May require modified versions to be distributed as the original version plus patches,
- No discrimination against persons or groups,
- No discrimination against fields of endeavour,
- All rights granted must flow through to and with redistributed versions,
- The license applies to the program as a whole and each of its components,
- The license must not restrict other software, thus permitting the distribution of open source and closed source software together.

What is free software?

Free software is defined in a slightly different manner by the Free Software Foundation. It is defined in terms of the freedoms that it provides:

- The freedom to run the program for any purpose,
- The freedom to study how the program works and adapt it to your needs, thus the requirement that the source code be provided,
- The freedom to redistribute copies so you can help your neighbour,
- The freedom to redistribute copies and the freedom to improve the program and release your improvements to the public, so that the whole community benefits, again necessitating the distribution of a source code.

In summary, free software is the freedom to run, copy, redistribute, study, change and improve software; all those attributes of copyright.

As Richard Stallman is fond of saying "it's free, as in free speech, not free beer!". Note that there is no requirement that you distribute the software at no cost. The definition of the Open Source Initiative probably encompasses a wider range of licences. We have heard at least one or two of those mentioned this morning such as, the BSD licence, the Apache Software Licence, the IBM Public Licence, the Common Public Licence and there are a whole range of these. And if in fact you were to look at the licence attributions in Red Hat Linux, which consists of about 2800 different software modules at this stage, you would find that probably half of those modules are licenced pursuant to the GNU General Public Licence, a sizeable number are licenced under the Berkley licence, the BSD, and the rest are licenced under various other open source licences or simply placed in the public domain.

As someone had said previously, in reality there are really few differences between Open Source and Free Software. The Open Source Initiative adopted the term Open Source because they thought the term 'free software' was confusing. It implied free as in no charge. I recently suggested to Eben Moglen, who is general counsel of the Free Software Foundation, that perhaps a better and more apt name for Free Software would be Freedom Software.

Why did Red Hat choose this open source approach? As stated by the Open Source Initiative 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, when one is used to the slow pace of conventional software development, seems astonishing. The Open Source community has learned that this rapid evolutionary process produces better software than the traditional closed source model, in which only a few programmers can see the source and where everybody else must blindly use an opaque block of bits.

RED HAT BUSINESS MODEL

I will take a second here to digress a little bit and talk some about Red Hat and our business model because one of the questions that I most commonly have to answer when I am at conferences is "How does Red Hat make money"? Red Hat is a combination of a lot of things. It enjoyed a great deal of insight by its founders, Bob Young and Marc Ewing. It enjoyed a great deal of just plain luck in that Red Hat went to the public markets at a time when the public markets were very hot and managed to not only make an initial public offering but also a secondary offering during that period of time, thus raising a great deal of capital for a company that was as young as we were. In fact in the fall of 1999, a few months after we went public, a business analyst said that Red Hat was a successful initial public offering in search of a business model. There was a degree of truth in it because at the time of the public offering we had one product, and that was the boxed set that you could buy at retail. We had no technical support model in place. We had just the early stages of our educational offerings. We had nothing in place to support developers. We did not have a concept behind us attracting other software vendors to migrate to Linux. None of those things existed.

In the three years since our public offering we have gone from one stream of revenue to over 20 streams of revenue today. We have a wider range of open source software offerings including different configurations of Linux for different applications. We have now penetrated the enterprise market, where large businesses are looking to shift their applications from proprietary software and hardware to open source software running on generic hardware, thereby creating significant monetary savings. The Red Hat Certified Engineer designation has become one of the most sought after designations in the IT industry. And I saw recently where we have won several awards and have been recognised for the work that we have done in that area.[6] In fact, for those of you who have a technical background, our Red Hat Certified Engineer designation has been shown to provide an 11% increase in earning power to those who received the designation.[7]

We have been leaders in terms of providing educational programs not just in live offerings, but also via web-based training. We developed a technical support model that is highly efficient in meeting the needs of our customers. We developed a developer support model to help other companies port their applications to Linux or develop applications for Linux. We have done work in the area of embedded Linux for embedded devices, such as cellular phones, internet devices that are freestanding, and the General Motor's Onstar system used for assisting vehicle owners.

---

[6] Red Hat Training News & Reviews, [http://www.redhat.com/training/news.html]

[7] John Roberts, "Key Certifications Grow in Value, Cisco Worth the Most, Red Hat Rising Fast" *CRN*, [http://www.crn.com/sections/special/ssurvey/ssurvey02.asp?ArticleID=35952] 18 December 2002.

We have expanded our Linux offering itself into a higher offering called 'Advanced Server' which we see as the future of Linux in the enterprise. It's receiving widespread kudos at the enterprise level. So we have vastly expanded what we are able to do in the last 15 months. We have gone from a company that was regularly losing money, although managing to not borrow cash resources a great deal, to one that is now approaching operating profitability and has, in fact, substantiated that the Open Source model can survive in business.

THE GNU GENERAL PUBLIC LICENCE

Let us turn now to licensing and run through a little bit about the GNU General Public Licence. The GNU General Public Licence is probably the most popular of the Open Source licences, and I would argue it is the backbone of the Open Source movement. It is made available by the Free Software Foundation, and I do want to acknowledge the contributions of Richard Stallman in terms of thought and leadership and Eben Moglen, their general counsel, in terms of legal leadership in this area. For those of you who are legal practitioners I will note for you that Eben, despite significant demands on his time as a fulltime Professor of Law at Columbia University, is very generous with his time and tries to be very responsive in the answering of questions. So if you have a tough nut to crack, do not hesitate to write to him there, and he will wish I had not said that, to volunteer his time.

The GNU General Public Licence provides the following rights:

- The right to copy and redistribute so long as you include a copyright notice and a disclaimer of warranties,
- You may charge for the cost of distribution and you may offer warranty protection for a fee,
- The right to make derivative works for your own use,
- The right to distribute derivative works so long as you meet certain conditions.

Those are:

1. To identify the work as being modified
2. That you licence it under the GPL
3. That you provide the licence information interactively if the program normally runs interactively.

This section and the obligation to licence under the GPL does not apply to works which are independent works distributed with the GPL work and which will simply run on GPL works:

- You may distribute the work only in executable forms so long as
  - the source code is distributed with the object code, or
  - the source code is offered by a written offer valid for a period of least 3 years to make the source code available for no more than the cost of distribution or
  - for non-commercial redistributions, where you have only received the object code, the notice you have received from the original distributor must be included
- Finally, you may not impose restrictions on any of these rights.

The GPL has no relevance to the case where a licensee chooses not to redistribute. For example, if you elect to adopt Red Hat Linux and make modifications to the Linux kernel, which is licenced pursuant to the GPL, so long as that modified kernel is only used within your organisation you are under no obligation to either licence that modification to another party or open source that modification. In this context, when an organisation is a corporate entity that corporate entity and all of its controlled subsidiaries are considered the same entity/organisation.

There are three common examples of the application of this redistribution principle:

1. users who use only GPL binaries as they would any other similar program,
2. users who modify GPL sources to handle internal issues,
3. and users who modify GPL sources and redistribute them for fun and/or profit.

In the first instance the simple use of GPL binaries has no impact on anything else you may be doing or running. A good example is the use of the GNU Emacs text editor which is GPL. The use of that text editor to open a file, edit that file, and save that file has no impact on the file so edited. Similarly, if that same file is an executable program which you can then compile using the GNU C compiler, again which is licenced under the GPL, the use of the compiler does not have any impact on the licence or ownership of the compiled program. Thus the normal use of GPL software in a commercial environment poses no extraordinary legal problems. The wide distribution of Linux operating system software over the last several years for use on commercial web and enterprise servers is ample evidence there is no legal reason to not use open source software if you happen to think it is better than the proprietary alternatives.

In the second case, the software was modified for internal use. That by definition confers to its users access to the internally modified sources. However, there is no requirement within the GPL that requires internal modifications to be distributed to another party or that the sources be disclosed to another party. Consequently, each user is free to modify GPL code for their own purposes without concern for disclosing trade secrets.

The final case is the one for which the GPL was really written. Users redistributing modified or unmodified versions of GPL source must obey the GPL's golden rule of not adding any downstream restrictions. To the extent that someone wants to profit from GPL software by using traditional proprietary licence restrictions, such restrictions will prove difficult if not impossible to enforce.

There are some finer points that I would like to walk through. For example, if the work that you have written is a database program intended to run on a GPL operating system like Linux, the mere distribution of the database program with the operating system on the same CD does not impose the GPL of the operating system to the database. Good examples of this include Red Hat's good friends at Oracle who are working hand in hand with us now to make Oracle available on Red Hat Linux. On the other hand, if modifications are made to that open source operating system in order to accommodate the database program, then those modifications are derivative works of the operating system and would need to be made available under the GPL. This imposition only applies to the derivative sources of the operating system and not to the database software itself.

Finally, the hardest cases come when proprietary software applications require modifications to the Open Source operating system in order to ultimately function. Where the proprietary software is so linked to the Open Source Operating System this has to raise issues of whether it is one piece of software or two. One must then examine issues of the type and nature of the modification or link, whether it is dynamic or static, and whether application interfaces have been used and the licence pertaining to those interfaces.

The wider the adoption of Open Source software the better understood the licencing principles pertaining to these specific cases are becoming. However, if you are not familiar with these issues, you are well advised to consult with a company that is familiar with the issues or discuss the specifics with a knowledgeable attorney, and I mentioned earlier Eben Moglen at the Free Software Foundation. In addition, you can find a discussion of some of these finer points on the FSF website.[8]

Some of you may be familiar with the Lesser GPL or have heard of it. The LGPL, although similar to the GPL in many respects, is intended primarily for use with libraries. It permits combining those libraries with other source code or linking with those libraries under certain conditions such that a derivative work is not created and the greater work is not subject to either the GPL or the LGPL.

[8] www.fsf.org

You also heard the term “viral” used in the context of GNU General Public Licence. Some of the proprietary developers have put out a lot of FUD on the GPL. For the non-techies, FUD stands for ‘fear, uncertainty and doubt’. Where you cannot beat somebody on the technical competency of the software, try to scare the public away from it. This is intended to undermine the confidence of parties who may be interested in building applications to run on GPL code. It has been asserted that because the GPL requires all derivative works to be licenced under the GPL, that the GPL code will in fact infect anything run with it. Well, as I pointed out before, that is simply not the case. The GPL only applies to derivative works. If you want to simply run your application on GPL code – no problem. There is no 'infection' if you will. [If you want to see Richard Stallman get excited, use the term “viral” in his presence.] It is also clearly not the case because the licencing obligation only applies for the modified code which **is redistributed**. The GPL makes very clear that it does not apply to independent programs that merely run with the GPL code.

One class of open source licenses, and the BSD would be a good example of this, does not tie itself to derivative works. The other class, and the GPL is the best example of this but the Open Public Licence and IBM Public Licence are also good examples, do tie themselves to derivative works, thus ensuring that all downstream works remain available to the community.

Why does Red Hat use the GNU General Public Licence as its principle licencing option? Although some of our software code is licenced under other Open Source licences, we principally use the GPL. We do so because of the obligation imposed on derivative works, thus extending the Open Source community and preventing other parties from simply ripping off the open source efforts. I have to tell you that this has been one of my great challenges in the time that I have been at Red Hat, and it has been a period of enlightenment for me as well. For that I have to credit our engineers, because despite time and time again having had debates especially over the first few years about our business model and about whether we do not need to have some proprietary offerings, we have consistently come back to the position that if we are willing to be an Open Source company that it means being an Open Source company. We will not start to integrate proprietary works into what we offer in order to provide some monetary benefit to ourselves. This has been a challenge for us, but I think in terms of our own corporate identity and who we are, it has been important for us to stay true to that vision.

Why do we not provide warranties or indemnities against infringement? Well, first the GPL expressly disclaims warranties and Red Hat in particular, is not in a position to simply assume that obligation on its own. It is not built into our pricing model as it is with proprietary software. There is no great pool of funds to provide those sorts of indemnities. In fact, often the software has been provided at no cost. At the same time there are benefits derived from the software being open and readily examined.

It is interesting to note that after more than ten years of use no GPL code has been challenged under claims of infringement. It is also interesting to note that the Microsoft standard end user licence for most of its products disclaims the warranty of non-infringement of intellectual property.

How is the GPL enforced? If you were to ask Eben Moglen he would say "to a large extent it is enforced by mediation". In fact he is often personally involved in carrying out that mediation. When the FSF is made aware of GPL violations they will contact the violating party and work with them to bring their application into compliance. And 99 times out of 100 the companies do want to be compliant. In all the years the FSF has been operating, this approach has been consistently effective in bringing about compliance.

It is again significant to note that after more than ten years of use the GPL itself has never been challenged in court. To my way of thinking, this is the best evidence of its durability. At the same time we are not so foolish as to believe that some parties may not elect to intentionally violate the GPL for purposes of prompting litigation. Keep in mind that, absent the rights granted under the GPL, there are rights under copyright and conditions that copyright imposes. A person obtaining GPL software has no rights under copyright law to make more than a single backup copy and to use that one copy. They have no right of redistribution. They have no right to make derivative works. Those are rights that can only be granted by the copyright owner, and they are granted by the GPL. If the GPL were held unenforceable, all of those rights would disappear.

As I have mentioned, there have been no court cases to date that have interpreted the GPL other than to sustain the fact that it is in fact an effective licence intended to restrict the manner in which the software is distributed. The Planetary Motion case has been mentioned by some: see *Planetary Motion, Inc. v. Techsplosion, Inc*.[9] This was a case that came down a few years back. A person involved with the litigation wrote me an email after it was decided saying that the GPL had been upheld by the Court. Well, if you read the case, the GPL was mentioned in the opinion but the case really had to do with trademark law. Similarly there is a court case pending in Federal District Court in Massachusetts at the present time involving MySQL and again this case largely involves trademark law. So to date we do not have specific US court interpretations of the GPL.

[9] 261 F.3d 1188 (11th Cir. 2001)

Chapter 2

Legal Issues Relating to Free and Open Source Software

PROFESSOR BRIAN FITZGERALD
BA (Griff) LLB (QUT) BCL (Oxon.) LLM (Harv) PhD (Griff)
Head of the School of Law
Queensland University of Technology, Australia

GRAHAM BASSETT
Barrister, Bank of New South Wales Chambers, Brisbane, Australia

---

1. INTRODUCTION

1.1 Background

This is a story about the models for distributing software and in particular the source (or human readable) code of the software. It is a story of many dimensions: law, politics, social planning, culture along with economics.

In the classic free software scenario embodied in the GNU General Public License (GPL) software source code is distributed in a manner that is open and free (as in speech not as in beer) allowing software developers (usually many hundreds, known broadly as the "hacker community") further down-the-line to modify and improve upon the initial software product. The initial distributor of the code controls its presentation and further dissemination through copyright and contract law (contractual software license). In general, the down the line developer and modifier is required to make source code of any derivative work that they *distribute* available for all to see. In this process copyright law is used to create a "copyleft" effect as opposed to a "copyright" effect by mandating that code should be open and free for all to use in innovation and development of software. By way of contrast, in a proprietary or closed distribution model source code is not released and can only be ascertained through decompilation or reverse engineering.

Software code is protected as expression in the form of a literary text under copyright law. Copyright law will protect the expression of an idea or facts but not the idea or facts themselves. Patent law and trademark law, amongst other things,

may also bestow legal rights in relation to software.[10] Once code is written it can be protected in copyright law for the life of the author plus fifty years in some countries and up to seventy years in other countries (USA, Europe). As a general rule if code is written in the course of employment then the employer will be the lawful owner of the copyright in that code. Software is generally not sold but distributed through software (contractual?) licenses. This has led some to say in relation to software and other informational goods that the "licence is the product". In the case of free and open source software the legal regime is built on the back of copyright in the original code along with the terms of the licence. Therefore the terms of the licence are crucial to understanding user and exploitation rights especially in a commercial setting.

1.2 Proprietary and Communal Software Licensing

Software licensing has two approaches — proprietarial and non-proprietarial.

Proprietary methods involve employing a team of programmers and tying them to a non-disclosure agreement. Cloistered for a period of time, they create, test and debug their code. Most importantly, copyright is claimed over the resulting code.[11] Software is marketed as a copyright license and defined as "any product we make available for license for a fee".[12] Bill Gates has made it clear that code is zealously guarded and presented in executable form only: "…a competitor who is free to review Microsoft's source code … will see the architecture, data structures, algorithms and other key aspects of the relevant Microsoft product. That will make it much easier to copy Microsoft's innovations, which is why commercial software vendors generally do not provide source code to rivals".[13]

---

[10] B. Fitzgerald, "Digital Property: The Ultimate Boundary?" (2001) 7 *Roger Williams University Law Review* 47; B. Fitzgerald and A. Fitzgerald, *Cyberlaw: Laws Relating to the Internet, Digital Intellectual Property and E Commerce* (2002) Lexis Butterworths, Sydney Australia; A. Fitzgerald, B. Fitzgerald, C Cifuentes and P Cook (eds.) *Going Digital 2000: Legal Issues for Electronic Commerce, Multimedia and the Internet* (2000) Prospect Publishing, Sydney.

[11] Software has been protected as a literary work under the US Copyright Act since 1980: *Lotus Development Corporation v Borland International Inc* 49 F. 3d 807 (1st Cir. 1995), and under the Australian *Copyright Act* since 1984: *Data Access Corporation v Powerflex Services Pty Ltd* [1999] HCA 49. Article 10(1) of the *Agreement on Trade-Related Aspects of Intellectual Property Rights* (TRIPS) Agreement, part of the World Trade Organisation Agreement of 1994 and binding on all members of the World Trade Organisation (WTO) provides that: "Computer programs, whether in source or object code, shall be protected as literary works under the Berne Convention (1971)". More recently software has been subject to a vast amount of patenting throughout the world: *Welcome Real-Time SA v Catuity Inc* [2001] FCA 445 (Australia); *State Street Bank & Trust Co v Signature Financial Group Inc* 149 F. 3d. 1368 (Fed. Cir. 1998) (USA).

[12] Microsoft Open License Agreement v 6.0, 1 October 2001, para [1] – applicable from 1 July 2002.

[13] *State of New York v Microsoft Corporation*, Direct Testimony of Bill Gates, 18 April 2002 [http://www.microsoft.com/presspass/trial/mswitness/2002/billgates/billgates.asp] at

Typically, proprietary licenses are sold under a Volume License Product Key (VPK) and the consumer is held liable for any unauthorized use of this key.[14] A customer can run the program which is defined as the capacity to copy, install, use, access or display the product for the number of copies authorised. A proprietary licensee may not "reverse engineer, decompile, or disassemble products except to the extent expressly permitted by applicable law".[15] This is contrary to the view that software diversity is best facilitated by reverse engineering.[16] Even so, a proprietary license recognizes that a copyright owner cannot require a licensee to enter a contract that is prohibited by statute, as is a contract to override reverse engineering rights in Australia. It is unclear in the US whether the right to reproduce a program in order to facilitate reverse engineering for the purpose of interoperability can be overridden by contract.[17] A licensee may not rent, lease, lend or host products.[18] In return, the user is offered a limited warranty that the product will "perform substantially in accordance with our user documentation" for a period up to ninety (90) days from first running the program.[19] Whether the final product is sold by shrinkwrap or clickwrap license, licensees are dependent on the vendor for upgrades and patches. Traditionally upgrades enabled a licensee to purchase modifications when, and, as they saw fit. Microsoft's Software Assurance scheme requires a user to buy an upgrade subscription as part of the license of a product.[20] Critics claim that this upgrade scheme applies a fee to the licensee even if no upgrade is provided in that period and this merely offers a "right to upgrade that previously existed without any requirement for advanced payment to preserve the right".[21]

Conversely, non-proprietarial software is created by communities of disparate developers for little commercial gain. Its benefits were propounded by Eric

---

[307] 20 April 2002; P Galli, "Microsoft Warns SEC of Open-Source Threat" http://www.eweek.com/article2/0,3959,857673,00.asp Note that a new Microsoft initiative proposes for security and other reasons that source code will distributed to government users under the Government Security Program: see P Calli "U.K. Adopts Microsoft's Security Program" <http://www.eweek.com/article2/0,3959,854839,00.asp.

[14] Note 3, para [4].

[15] Note 3, para [7].

[16] B. Fitzgerald, C. Cifuentes, A. Fitzgerald and M. Lehmann, "Innovation, Software and Reverse Engineering" (2001) 18 *Santa Clara Computer And High Technology Law Journal* 121; B. Fitzgerald "Intellectual Property Rights in Digital Architecture (including Software): The Question of Digital Diversity?" [2001] *EIPR* 121.

[17] See s 105 *Uniform Computer Information Transactions Act* (UCITA); *Bowers v Baystate Technologies Inc* 64 USPQ 2d. 1065 (Fed. Cir 2002).

[18] Note 3.

[19] Note 3, para [9a].

[20] Note 3, para [11].

[21] A NZ company has made a formal complaint about the impact of this new 'software-as-service' paradigm. See "Complaint to the Commerce Commission by Infraserv Limited as to certain anti-competitive behaviour of Microsoft NZ Limited", [www.clendons.co.nz], 9 May 2002.

Raymond in his work *The Cathedral and the Bazaar*, first written in May 1997.[22] He compares the commercial (non-free) development of software to the building of large cathedrals, "carefully crafted by individual wizards or small bands of mages working in splendid isolation, with no beta to be released before its time".[23] Slow to react, and with too few participants, the cathedral approach is not as effective as that of the bazaar:

> … release early and often, delegate everything you can, be open to the point of promiscuity … No quiet, reverent cathedral-building here – rather, the Linux community seemed to resemble a great babbling bazaar of differing agendas and approaches (aptly symbolized by the Linux archive sites, who'd take submissions from anyone) out of which a coherent and stable system could seemingly emerge only by a succession of miracles.[24]

Raymond identified 17 features of the non-proprietarial system that contributed to its successful creation of software such as the alternate operating system GNU/Linux.[25] The most important are:

- Good programmers know what to write. Great ones know what to rewrite and reuse.
- Many programmers build on the efforts of others rather than doing things from scratch.
- The best programs are written by reacting to a need perceived by programmers.
- Treating users as co-developers allows bugs to be identified quickly and more effectively.
- Such communities have a scalability of reaction to a perceived problem that commercial developers even as big as Microsoft have come to realize.[26] Raymond asserts: "[G]iven enough eyeballs, all bugs are shallow".[27]
- Programs are best released early and often and feedback sought from users.

Raymond argues developers in such communities are not motivated by commercial gain. An internal market of reputation exists. The principal role of the project leader is to facilitate "egoless programming". On describing the role of a project leader in such a distributed format, he quotes the words of a Russian anarchist, Kropotkin:

---

[22] Eric Raymond, *The Cathedral and the Bazaar*, version 2, 24 August 2000, <http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/index.html> (27 July 2001).
[23] Note 13.
[24] Note 13.
[25] For a brief overview of the development of GNU/Linux see section 1.3.1 of this paper.
[26] In two leaked memos Microsoft admitted to the benefits of non-proprietarial development methods at Linux and proffered whether such development could be slowed by court challenges: Bob Trott, "Microsoft Pondering Legal Challenge to Linux", *CNN.com*, November 1998, <http://www.cnn.com/TECH/computing/9811/06/linux.threat.idg/>, (22 November 2001).
[27] Note 13.

> I began to appreciate the difference between acting on the principle of command and discipline and acting on the principle of common understanding. The former works admirably in a military parade, but it is worth nothing where real life is concerned, and the aim can be achieved only through the severe efforts of many converging wills.[28]

Raymond claims community members are driven by "egoboo" – the enhancement to self-esteem that results from successful participation in the group.[29] The free community can bring more rapid attention to a problem whereas the proprietary closed-source responses are "frequently as late as they are disappointing".[30] "In the world of cheap PCs and fast Internet links we find pretty consistently that the only really limiting resource is skilled attention."[31] Critically, developers should have access to the source code of a program thus enabling modification and distribution with limited obligations to the licensor. It is source code that "links computers and humans. To understand how a program runs; to be able to tinker with it and change it; to extend a program or link it to another — to do any of these things with a program requires some access to the source".[32]

Free and open source development also has supporters in the commercial market place. Equipment manufacturers are keen supporters of open source software that will make their machines interoperable and broaden potential market share. They might use free or open code in ROM chips which are executed when a computer starts up and dictates the range of activities it can perform. Interoperability is needed between these ROM chips and any software to be added later. For example, a manufacturer of computers may wish to be operable with Sun's Star Office application and would use any open source or free code that could enhance such operability. In order to have their machines used as Web servers they would wish to be operable with all ranges of web software on the market. A hardware company that could sell a machine with a free or open source operating system may be able to increase its profit margins when it does not have to pay mandatory license fees to a particular supplier. Recognising quality, in 1998 IBM stopped putting out its own server product with its machines and adopted Apache, an open source server product with over 80% market share.

---

[28] Note 13.
[29] Note 13.
[30] Patrick K Bobko, "Open-Source Software and the Demise of Copyright" (2001) 27 *Rutgers Computer & Tech. L.J.* 51 at 79.
[31] Note 13.
[32] Lawrence Lessig, *The Future of Ideas*: *The Fate of the Commons in a Connected World* (2001) Random House, NY, p 50.

Governments, too, recognise the benefits of open source development.[33] A recent study by Mitre Corporation on behalf of the US Department of Defence was cautiously optimistic concluding open source "encourages significant software development and code re-use, can provide important economic benefits, and has the potential for especially large direct and indirect cost savings for military systems that require large deployments of costly software products".[34] Taiwan has an open source project supported by the National Science Council and Ministry of Education. It is examining use of open source products to save royalty payments for office software in government agencies and schools.[35] Due to the high regard for privacy considerations in Europe, the German government is supporting an open source project, GnuPG, to reduce reliance on proprietary privacy enhancing code such as PGP.[36] The Linux community has entered a cooperative project with the Software Research Institute of the Chinese Academy of Sciences and NewMargin Venture Capital, a venture arm of the Chinese government called RedFlag. Initially, it developed a localised operating system for servers but now incorporates developments for PC systems, PDAs, and China's computerized lottery system.[37] The Peruvian parliament has a Bill before it to mandate use of open source products in government offices. Peruvian Congressman, David Villanueva Nuñez, circulated a letter to Microsoft on the Internet that sparked much debate on the relative merits of free and open code as opposed to proprietary development.[38]

The clash between nonproprietary and proprietary forms of development is a political, social and economic struggle. "Software pervades modern society; it can be found in almost every product. So naturally, if only a few people control software, their power increases and restricts users' freedom."[39] Development of such 'gift cultures' can mitigate "worrisome concentrations of corporate power in

---

[33] For an article overviewing international legal efforts to utilise open source code see: Paul Festa, "Governments push open-source software", *CNET News.com*, 29 August 2001, [http://news.com.com/2100-1001-272299.html?legacy] 19 July 2002.

[34] Carolyn A. Kenwood, "A Business Case Study of Open Source Software", The MITRE Corporation, July 2001, [http://www.mitre.org/support/papers/tech_papers_01/kenwood_software/index.shtml] 19 July 2002. pxxv.

35 Tiffany Kary, "Taiwan opens door to open source", *ZDNet News*, June 4, 2002 [http://zdnet.com.com/2100-1104-931885.html] 19 July 2002.

[36] The GNU Privacy Guard, [http://www.gnupg.org/].

[37] [http://www.redflag-linux.com/].

[38] Thomas C Greene, "MS in Peruvian open-source nightmare", 5 June 2002, [http://www.theregister.co.uk/content/4/25157.html] 17 July 2002.

[39] Shawn W Potter, "Opening Up to Open Source", (2000) 6 *Rich J.L. & Tech* 24, <http://www.richmond.edu/jolt/v6i5/article3.html>, (18 August 2001); B Fitzgerald, "Intellectual Property Rights in Digital Architecture (including Software): The Question of Digital Diversity?" [2001] *EIPR* 121.

the software industry [by disdaining] those who seek to financially profit from the community's shared body of knowledge".[40]

This does not mean non-proprietary development groups are united. They differ over how to best maintain the communal aspects of software development. Since the stock market 'tech wreck', non-proprietary groups have accelerated their split into two main camps — the free software and open source movements. Their essential difference lies in their approaches to the commercialization of non-proprietary code.

1.3 Free Software Foundation (FSF) and Copy Left

"Free software does not mean that the software is free, as in requiring no payment. When I speak of free software, I'm referring to freedom, not price. So think of free speech, not free beer."[41] Thus asserts Richard Stallman, the founder of the Free Software Foundation. Software is not free because it has no price, it is free because it contains values that enhance liberty for users and programmers. Stallman applies four strict criteria to maintain free values in software:

1. The freedom to run the program, for any purpose (freedom 0).
2. The freedom to study how the program works, and adapt it to your needs (freedom 1). Access to the source code is a precondition for this.
3. The freedom to redistribute copies so you can help your neighbour (freedom 2).
4. The freedom to improve the program, and release your improvements to the public, so that the whole community benefits (freedom 3). Access to the source code is a precondition for this.[42]

---

[40] David Bollier, "The Power of Openness, Why Citizens, Education, Government and Business Should Care about the Coming Revolution in OpenSource Code Software: A Critique and A proposal for the H_20 project", paper for the *Berkman Center for Internet and Society",* Harvard University, 10 March 1999, <http://eon.law.harvard.edu/opencode/h20/> (23 July 2001); B. Fitzgerald, "Software as Discourse: The Power of Intellectual Property in Digital Architecture" (2000) 18 *Cardozo Journal of Arts and Entertainment Law Journal* 337.

[41] Stallman Richard M, "Free Software: Freedom and Cooperation", Speech at New York University, New York, 29 May 2001 <http://www.gnu.org/events/rms-nyu-2001-transcript.txt> (27 August 2001). On the power of free software models to enhance digital diversity consider: B Fitzgerald, "Intellectual Property Rights in Digital Architecture (including Software): The Question of Digital Diversity?" [2001] *EIPR* 121; B. Fitzgerald, "Software as Discourse: The Power of Intellectual Property in Digital Architecture" (2000) 18 *Cardozo Journal of Arts and Entertainment Law Journal* 337.

[42] "The Free Software Definition", Updated 27 October 2001, <http://www.fsf.org/philosophy/free-sw.html> (23 July 2002).

1.3.1 GNU/Linux

Stallman realised the advantage of accessing source code when he tried to change a program given to him by Xerox so it would run on printers in his MIT lab. Xerox did not make the source code available and instructed an employee not to give Stallman copies of the software. Comparing the sharing of programs to the sharing of recipes (and the improvement of programs and recipes by changing the source), he strongly criticizes proprietary control of software:

> So imagine what it would be like if recipes were packaged inside black boxes. You couldn't see what ingredients they're using, let alone change them, and imagine if you made a copy for a friend, they would call you a pirate and try to put you in prison for years. That world would create tremendous outrage from all the people who are used to sharing recipes. But that is exactly what the world of proprietary software is like. A world in which common decency towards other people is prohibited or prevented.[43]

Free software developers often contribute to a program by accident and share source code on some "intellectual commons",[44] usually the Internet.[45] One module can be used to integrate with a need of another developer. Thus software progresses incrementally.

When broken up due to anti-trust findings against it in 1984, AT&T tried to license use of its previously free Unix code. In reaction to this, Stallman launched a project called GNU – a recursive acronym meaning "GNU's not Unix". The aim was to create tools to build an operating system and then to produce such a system called GNU OS. By the end of the 1980s tools such as a compiler had been developed but the project slackened as the kernel for GNU OS was being formed. In 1991, Stallman found by accident the Linux module developed by Linus Torvalds in Finland to add to his GNU program after his attention was drawn to it by other free developers who had seen parts on the Internet.[46] Torvalds joined with Stallman and the GNU/Linux operating system emerged. This system was distributed with its source code.

In '91 only Macintosh had advanced beyond a command-driven interface for operating systems. Such a system of complicated commands had to be mastered chilling the proliferation of computers to novice and irregular users. How many wanted to remember a command like 'copy *.* A:' in order to copy all files to a

---

[43] Note 32.
[44] For the enhanced power code writers have in the cyberspatial intellectual commons see: Lawrence Lessig, "Symposium: Key Address: Commons and Code" (1999) *9 Fordham I. P., Media & Ent. L. J* 405 at 410.
[45] For example, Apache is a widely used open source, web server program: <http://httpd.apache.org/dist/> (23 November 2001).
[46] Note 32. The history of this connection is best covered in the book by Sam Williams mentioned in Note 38.

floppy disk from a hard drive? During '91 Microsoft developed a serious graphical user interface (GUI) for an operating system not reliant on a DOS command driven system. The methods used by Microsoft to develop and ensure consumer loyalty to this system they developed and propertized is the subject of an extensive series of ongoing anti-trust cases. The GNU/Linux system has also met this challenge by adopting contributions of groups such as Samba to produce GUI interfaces for file and print servers.[47]

The result is that Linux now represents an operating system with significant profile in the community, so much so that it is said to threaten the dominance of MS Windows. A recent statistical study shows some comparative findings:

- Reliability: In a ten-month test for reliability run by ZDNet, NT servers crashed an average of once every six weeks, the GNU/Linux server*s* never went down,
- Security: Insurance companies covering "hacking" incidents have begun charging clients 5 to 15 percent more where Microsoft's Windows NT software is employed instead of Unix or GNU/Linux, in Internet operations.[48]

1.3.2 General Public License (GPL)

A licensing system that promoted sharing and innovation was critical in the development of GNU/Linux. The free software movement is concerned with colonization by commercial (non-free) groups incorporating free code into their developments. Any copyright taken out over the resultant program effectively privatizes the free code used. Furthermore, non-free developers have freeloaded by not contributing anything back to the free community. To counter this, Stallman introduced the GNU General Public License (GPL). The GPL covers the initial program and "any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language".[49]

Stallman places the GPL in a direct commercial and political context called 'Copyleft':

> To copyleft a program, we first state that it is copyrighted; then we add distribution terms, which are a legal instrument that gives everyone the rights to use, modify, and redistribute the program's code *or any program derived from it*

---

[47] For a detailed overview of the history of GNU/Linux see: G. Moody, *Rebel Code: Linux and the Open Source Revolution*, (2001) Penguin Books, NY, USA. See also: Lawrence Lessig, *The Future of Ideas*: *The Fate of the Commons in a Connected World* (2001) Random House, NY, p50ff; Sam Williams, *Free as in Freedom: Richard Stallman's Crusade for Free Software* (2002) O'Reilly, San Francisco, Chapter 11.

[48] David A. Wheeler, "Why Open Source Software / Free Software (OSS/FS)? Look at the Numbers!" [http://www.dwheeler.com/oss_fs_why.html] 23 April 2002.

[49] "The General Public License (GPL)", Version 2, June 1991, <http://www.opensource.org/licenses/gpl-license.html>, 19 August 2001.

but only if the distribution terms are unchanged. Thus, the code and the freedoms become legally inseparable.[50]

It is a powerful license. By mixing GPL'd code with new code in a derivative work the obligation arises to make all the source code known or "free". Consequently, a commercial developer who takes free code under a GPL license into the code of their product is obliged to make the source code of the entire product available. Stallman argues this ensures 'freedom three' is maintained and the whole community benefits. The GPL "actually has the strength to say no to people who would be parasites on our community".[51]

1.4 The Open Source Movement

The open source movement is a non-profit organization. Its leading proponent, Eric Raymond, has conceptualized business models enabling commercial exploitation of open source programs.[52] Programs distributed with the Open Source Certified trademark (OSI Certified)[53] are published on an approved list of licenses[54] that conform to the open source definition.[55] The main elements of such licenses are:

- Free redistribution so that a party may not charge a fee or royalty for the program unless it is a component of an aggregate software distribution containing programs from several different sources.
- The license shall not require a royalty or other fee for such sale,
- The program must include source code, and must allow distribution in source code as well as compiled form. If a program is not distributed with source code (eg Apache license) 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,
- Derived works and modifications must be allowed and be capable of distribution under the same terms as the original license,
- The license must maintain the integrity of the authors code by guaranteeing that source be readily available, but may require that it be distributed as pristine base sources plus patches. In this way, "unofficial" changes can be made available but readily distinguished from the base source,

---

[50] "What is Copyleft?", Updated 5 November 2001, <http://www.gnu.org/copyleft/copyleft.html> 24 November 2001.
[51] Richard M Stallman, Note 32.
[52] These include loss leader; widget frosting; give away recipe/open restaurant; accessorizing; free the future, sell the present; free the software, sell the brand; free the software, sell the content. Potter Shane W, "Opening UP to Open Source" (2000) 6 *Rich. J.L &Tech* 24
[53] Open Source.Org, Revised 30 April 2001, <http://www.opensource.org/docs/certification_mark.html> (24 November 2001).
[54] Open Source.Org, <http://www.opensource.org/licenses/index.html>, (24 November 2001).
[55] Open Source.Org, Version 1.9, <http://www.opensource.org/docs/definition.html>, (20 July 2002).

- The license must not discriminate against any person, group of persons or fields of endeavour,
- The license must not discriminate against fields of endeavour,
- The rights attached to the program must not require entry to some other form of license or agreement such as a non-disclosure agreement,
- The rights attached to the program must not depend on the program's being part of a particular software distribution,
- The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.[56]

1.5 Tension between Open Source and Free Software

It could appear that the difference between open source and free software is minimal but the move to embrace the commercial market by open source developers has led the free software movement to clearly differentiate themselves. The distinction is over the implications of the GPL. Copyleft software maintains freedom for all developers (and consequently users). It requires a licensee to give back under the terms of the GPL the source code of any changes or modifications. Open source software will allow non-free versions to be made. For example, the Apache license allows a work to be distributed with or without modifications in source or binary form. The licensee can make changes without a requirement to share them provided the name of the derivative work is changed. The Berkeley Software Distribution (BSD) License contains no obligation to disclose source code of modifications when distributing a derivative work.[57]

Open source developers can use free software to add to proprietary software so a system can become a combination. While acknowledging that the real enemy is proprietary software companies, FSF are wary of open source groups who mix free and non-copylefted software. "In effect, these companies seek to gain the favourable cachet of 'open source' for their proprietary software products – even though those are not 'open source software' – because they have some relationship to free software or because the same company also maintains some free software".[58]

The tension is exacerbated by what many call the 'viral' nature of free GPL-licensed (GPL'd) code.

---

[56] Ibid.
[57] For more on the differences between GPL free software and open source see: Joe Barr, "Live and let license" ITworld.com, 23 May 2001, [http://www.itworld.com/AppDev/350/LWD010523vcontrol4/]; Larry Rosen, "Which open source license should I use for my software?" [http://www.rosenlaw.com/html/GL5.pdf].
[58] Gnu.Org, "Why 'Free Software is better than 'Open Source'", <http://www.gnu.org/philosophy/free-software-for-freedom.html>, (13 November 2001).

2. TABLE 1 - BASIC CLAUSES OF SOME FREE AND OPEN SOURCE LICENSES

2.1 The GNU General Public License (GPL)[59]

Definition of source code (Clause 3) – "the preferred form of the work for making modifications to it". Includes for all modules, interface definitions files, scripts for compilation and installation of the executable.

Program or "work based on the Program" = the Program or any derivative work under copyright law "a work containing the Program or a portion of it, either verbatim or with modifications and / or translated into another language". (Clause 0)

Aggregation of other works not based on the Program (eg stored together or shared on a server) does not bring other work under this license.

Clause	**Permits**	**Obliges**
1	Copy and distribution of verbatim copies Charging fees for: • Act of transferring copies • Offer warranty protection in exchange for a fee	"Conspicuously and appropriately publish" on each copy a copyright notice Disclaimer of warranty Pass on a copy of this license with the copy Keep intact any current notices in program
2	Make modifications "thus forming a work based on the program" Copy and distribute modifications	Place prominent notices in modified files of your changes and date thereof Allow any work "that in whole or in part contains or is derived from the Program any part thereof" to be licensed "as a whole" for no charge to all third parities under the terms of this License.
3	Copy and distribute work as object code or in executable form	Accompany this with complete machine-readable source code under license requirements of Clause 1 and 2 OR accompany it with written offer for up to 3 years to give any third party at a fee for no more than costs the source code under license requirements of Clause 1 and 2
5		Modification or distribution of the Program constitutes acceptance of the license terms
7		Legal obligation under other intellectual property rights prevent you from distributing the Program
11	No expressed or implied warranty, not even with regard to merchantability or fitness for purpose	

[59] www.opensource.org/licenses/gpl-license.html.

	Entire risk and cost for defects is borne by licensee	
12	Licensor is immune from damages arising from Program unless stipulated in writing	

2.2 GNU Lesser Public License[60]

Notes: this license is for software libraries. A software library is a "collection of software functions and / or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables. Yet when a program is linked with a library the result is [maybe][61] a derivative work. Under GPL this would mean the entire work would have to be made free. Compatibility of the library with other software that may be proprietarial is paramount. To avoid that whole work coming under GPL requirements the aim of this license is to "permit linking those libraries into non-free programs".

For example, proprietarial developers may wish to use a 'free' library that has become a de-facto standard such as many of the Linux standards. This license would allow this software standard to be incorporated into a non-free product.

Terms
Work based on the library = code derived from the library
Work that uses the library = code must be combined with the library in order to run

1	Copy and distribution of verbatim copies of the library's complete source code	"Conspicuously and appropriately publish" on each copy a copyright notice Disclaimer of warranty Pass on a copy of this license with the copy

[60] www.opensource.org/licenses/lgpl-license.html.

[61] During dissemination for comment of an earlier draft of this article, an e-mail debate took place over the viral nature of the GPL. A concern was raised that linking a core program to a library would make the core program a derivative of that library. It would appear, however, that without any modification of the library, mere linking would only create a 'redistribution' of that library, rather than a 'derivative' work. There is still some debate about this issue – Richard Stallman, of the Free Software Foundation, believes that "using the ordinary GPL for a library makes it available only for free programs" (Richard Stallman, *Why you shouldn't use the Library GPL for your next library* (1999) GNU Project <http://www.gnu.org/licenses/why-not-lgpl.html> at 22 September 2003). However, it is suggested that the more persuasive view is that a program will be a derivative of another if "the source code of the original program was used, modified, translated or otherwise changed in any way to create the new program", but not by "linking to library programs that were designed and intended to be used as library programs" or by utilising "plugins and device drivers that are designed to be linked from off-the-shelf, unmodified, programs" (Larry Rosen, *Geek Law: Derivative Works* (2003) Linux Journal <http://www.linuxjournal.com/article.php?sid=6366> at 22 September 2003). The distinction is important because redistribution of a library licensed under the GPL, rather than distribution of a derivative of that library, will not impose the restriction that the core program also be released under the GPL.

		Keep intact any current notices in program
2	Modify the library or any portion of it Copy and distribute modifications	Modified works must themselves be a software library and copied and/or distributed with notifications for free to third parties
3	Can apply terms of the GNU GPL to a derivative work not part of a library that uses a portion of code from the library	
4	Distribution of copies of the library or portions or derivative works must be accompanied by Source code	
5	Work that Uses the Library Program that contains none of the library but is designed to work with it is outside scope of license as are those that have only minimalist links to the library such as numerical parameters, data structure layouts, inline functions of ten lines or less.	Linking a work that uses the library with the library creates an executable that is a derivative of the library. This executable is covered by the License

2.3 BSD[62] and MIT License

	BSD Redistribute and use program in source or binary form with or without modification MIT No limits on rights to use, copy, modify, merge, publish, distribute, sublicense, and / or sell copies of the software	Redistribution of source code and binary form to contain copyright notices and disclaimer as to warranty only, license silent on making source code for derivative works available Need to publish disclaimer as to warranty

2.4 The Artistic License[63]

Package = collection of files distributed by copyright holder, and derivatives of that collection of files

Standard Version = a program package that has not been modified or has been modified in accordance with wishes of the copyright holder

1	Copy and distribute verbatim copies of the Standard Version of the Package	

[62] www.opensource.org/licenses/bsd-license.html.
[63] http://www.opensource.org/licenses/artistic-license.php.

2	Can apply fixes and other modifications derived from public domain or from the copyright holder of the Package. Package so modified becomes the standard version	
3 /4	Can further modify / distribute package	Notices in each changed file and must do ONE of the following: 1. Place modifications in a public domain 2. Use modified package only in own organization 3. Make other distribution arrangements with copyright holder 4. Rename executables not in standard version and provide documentation of how they differ to standard
5	Charge fees you choose for support of package but not the Package itself Can distribute Package in aggregate with other (possibly commercial) packages	May not advertise the package as a product of your own

2.5 Sun Industry Standards Source License (SISSL)[64]

Initial developer = individual or entity identified as initial developer in source code notice
Larger work = a work which combines original code or portions thereof with code not covered by terms of this license
Original code = source code of software
Contributor version = combination of original code and modifications made by that contributor

2.1(a) – (d)	Initial Developer Grant World wide, royalty free, non exclusive license subject to third party intellectual property claims to use, reproduce, modify, display, perform, sublicense and distribute original code with or without modifications and / or as part of a larger work A patent claim to sell and offer for sale the original code.	No patent license for code portions taken from original code or modifications to it.
3.1		Distribution of licensee's modified code must comply with all requirements of the Sun standards

[64] www.opensource.org/licenses/sisslpl.html.

		body. If they do not meet requirements must publish deviations from standards and offer in source code form under this license
3.2	Can charge fee for warranty support indemnity liability obligations for modified works but not Initial Developer	
3.3	Distribute any executable and source versions of modifications under license of own choice	Terms which differ from this license are not offered by Initial Developer and indemnify initial developer from any liability
3.4	Can combine original code with other code to create a larger work and distribute as a single product	

2.6 Mozilla Public License Version 1.0[65]

Contributor = each entity that creates or contributes to the creation of Modifications
Contributor version = original code, prior modifications and modifications of particular contributor

2.1	(a) Initial Developer grants world-wide, royalty-free, non-exclusive license (subject to third party intellectual property claims) to use, reproduce, modify, display, perform, sublicense and distribute (b) Utilize patents of original developer to the extent necessary to use original code	
2.2	Each contributor to give rights under 2.1 (a)-(b) above	
3.2		Source code versions of modifications must be made available on same media as executable version of program or via an accepted electronic distribution mechanism (if by EDM must be available for 12 months after date it initially became available or at least 6 months after a subsequent version became available)

[65] www.opensource.org/licenses/mozilla1.0.html.

2.7 Apache License Redistribute source code and binary forms with or without modification		
1-2	Redistributions in source or binary form must include copyright notice, disclaimers and list of conditions in license	
3.		End-user documentation included with the redistribution, if any, must include acknowledgment of Apache
4-5		May not use Apache name in derivative works or to endorse any derivative works
2.8 Open Software License[66] "Source Code" = the preferred form of the Original Work for making modifications to it and all available documentation describing how to modify the Original Work		
1	Licensor grants a world-wide, royalty-free, non-exclusive, perpetual, sublicenseable license to reproduce, distribute, perform, display and create and distribute derivative works	Copies of original work or derivative works distributed must be licensed under the Open Software License
2	Licensor grants a license to make, use, sell and offer for sale the original work and derivative works under any patent claims owned or controlled by the licensor and embodied in the original work	
3		Machine readable source code and documentation must be provided for everything released. Source may be published in an inexpensive and convenient information repository
4		The names and trademarks of the licensor or any contributors may not be used to endorse derivative works without consent.

[66] www.opensource.org/licenses/osl-2.0.php.

		No rights to trademarks, copyrights, trade secrets, patents or other intellectual property except as explicitly stated.
6		All copyright, trademark, patent and attribution notices in the original work must be retained in any derivatives. Source of derivative must contain a prominent attribution notice.
7-8		Warranty that copyright and patent rights licensed are granted by the licensor or are properly sublicensed with the permission of the contributor(s). Disclaimer of warranty and limitation of liability.
10		License is revoked immediately upon starting an action, including a cross-claim or counterclaim, for patent infringement, either (i) against the licensor for any software patent; or (ii) against any entity with respect to a patent applicable to the original work.

2.9 Academic Free License[67]

"Source Code" = the preferred form of the Original Work for making modifications to it and all available documentation describing how to modify the Original Work

1	Licensor grants a world-wide, royalty-free, non-exclusive, perpetual, sublicenseable license to reproduce, distribute, perform, display and create and distribute derivative works	
2	Licensor grants a license to make, use, sell and offer for sale the original work and derivative works under any patent claims owned or controlled by the licensor and embodied in the original work	

[67] www.opensource.org/licenses/afl-2.0.php.

3		Machine readable source code and documentation must be provided when the original work is distributed. No obligation is placed upon derivative works.
4		The names and trademarks of the licensor or any contributors may not be used to endorse derivative works without consent. No rights to trademarks, copyrights, trade secrets, patents or other intellectual property except as explicitly stated.
6		All copyright, trademark, patent and attribution notices in the original work must be retained in any derivatives. Source of derivative must contain a prominent attribution notice.
7-8		Warranty that copyright and patent rights licensed are granted by the licensor or are properly sublicensed with the permission of the contributor(s). Disclaimer of warranty and limitation of liability.
10		License is revoked immediately upon starting an action, including a cross-claim or counterclaim, for patent infringement, either (iii) against the licensor for any software patent; or (iv) against any entity with respect to a patent applicable to the original work.

3. SOME LEGAL ISSUES IN FREE AND OPEN SOURCE LICENSED SOFTWARE

3.1 The "Viral" Nature of Free Software

The GPL aims to "control the distribution of derivative or collective works". Clause 2 states that you can form a new work based on the original program provided that such a derivative work is itself licensed to all third parties at no charge under the terms of the GPL. Source code must be supplied. It is not aimed at claiming rights over entirely new works. The GPL refers to a derivative work as defined in copyright law.[68] In US copyright law[69] a derivative work must be based

[68] GPL, Note 40, Clause 0. Nonetheless there still appears to be some uncertainty as to what this means and as to whether this is the only type of derivative work contemplated by the GPL. Is any work that employs GPL'd code regardless of whether it would be a derivative work under US copyright law – a derivative work in the eyes of the GPL?

on one or more pre-existing works, and must recast, transform or adapt an original work.[70] To be a derivative work the licensee must change the code in the original GPL'd work (pre-existing work). The copyright owner of the pre-existing work has the right to authorise preparation of derivative works.[71] Copyright subsists for the author of the derivative work[72] but only to the extent of the material added to the pre-existing work.[73] In addition, copyright for the derivative work "does not extend to any part of the work in which such material has been used unlawfully".[74]

Here lies an interesting issue. The copyright statute gives the author of the derivative work the right to control copyright in the added (non pre-existing) parts of a derivative work.[75] The GPL insists they must reveal such changes. In this sense the GPL appears to override the rights given under the copyright statute to control the added (non pre-existing) parts of the derivative work. However, for the author of a derivative work to claim copyright under US law they will need to show they have not used the pre-existing work unlawfully.[76] Therefore an author of a derivative program will need to rely on the permission to use code granted in the GPL (which also contains an obligation to disclose source code) in order to assert copyright in the derivative work, thereby relinquishing complete control over the added (non pre-existing) parts of the derivative work. Although, in an instance where the use of GPL'd code is a lawful act of fair use under s 107 of the US

---

[69] Australian copyright legislation does not use the term "derivative works". A copyright holder has the exclusive right to make adaptations of a work under ss31 (1) (a) (vi) of the Australian *Copyright Act 1968*. For software purposes, an adaptation is a "version of the work (whether or not in the language, code or notation in which the work was originally expressed) not being a reproduction of the work": s 10 *Copyright Act 1968*. See also B Sookman, *Computer, Internet and Electronic Commerce Law* (1991) Carswell Canada 3-210; *Vault Corp v Quaid Software Ltd* 847 F.2d. 255 (5th Cir. 1988). Under s 31 (1) (a) (i) Copyright Act 1968 it is unlawful to reproduce the whole or a substantial part of a work without permission of the copyright owner: s 14 (1) *Copyright Act 1968*; *Data Access Corporation v Powerflex Services Pty Ltd* [1999] HCA 49.

[70] *Copyright Act*, (USA) 17 USC § 101.

[71] Note 61, § 106(2).

[72] Note 61, § 103(a).

[73] Note 61, § 103(b).

[74] Note 61, § 103(a). Contrast the Australian position: *A-One Accessory Imports Pty Ltd v Off Roads Imports Pty Ltd (No 2)* (1996) 34 IPR 332; S. Ricketson, *The Law of Intellectual Property* 2nd ed. Volume 1 (1999) Law Book Co., Sydney, 116-120.

[75] Note 61, § 103(a).

[76] Ibid. A further issue is the extent to which the unlawful user prevents copyright arising in any part of the later work: M. Nimmer and D. Nimmer, *Nimmer on Copyright* Lexis Nexis, Chapter 3. Cf. the Australian law: *A-One Accessory Imports Pty Ltd v Off Roads Imports Pty Ltd (No 2)* (1996) 34 IPR 332, which suggests copyright can be claimed and protected in an adaptation unlawfully embodying a pre-existing work. This may impact on the legal significance of the GPL under Australian law.

Copyright Act the need to gain permission pursuant to the GPL is removed.[77] Does this mean there is no obligation to disclose source code in this instance?

It is these types of uncertainties that have led non-free developers to fear their source code might need to be revealed by any contact with GPL licensed software. Compounding these concerns, the GPL is silent on length of license term.[78]

With one exception to be covered later in this paper, the GPL has not been tested in court. In order to avoid copyright infringement and the need to rely on the GPL, the down-the-line developer who wishes to distribute software will need to show that their code is an independent work and that it does not infringe the copyright owner's exclusive right to reproduce or prepare a derivative work. What does current law say is an infringement of the exclusive right of the copyright owner to prepare a derivative work under s106 US *Copyright Act*? There is not a plethora of cases testing this issue with regard to software. For an infringement, there must be substantial similarity in the "total concept and feel" of a derivative work in comparison to the underlying work. "The little available authority suggests that a work is not derivative unless it has been substantially copied from the prior work."[79] Yet, in another case, digital images that were "touched up or modified selections" of original works were not found to make the later work an infringing derivative.[80] Most importantly for free and open source developers, infringement was not found in a later work that was an improvement of a Nintendo computer game.[81] The later work allowed a user to add lives to a game character, increase the speed of its movements and empowered it to float over obstacles. However, in another case, a company that downloaded game levels created by users utilising the 'build' utility provided in the game and burnt these to a CD for commercial gain was found to have created an infringing derivative work.[82]

---

[77] See further: S. McJohn, "The Paradoxes of Free Software" (2000) 9 *Geo. Mason L. Rev.* 25.

[78] See further *Uniform Computer Information Transaction Act (UCITA)* s308(1) & (2). Updated August 10 2001, <http://www.ucitaonline.com/ucita.html> (23 November 2001). UCITA is a model law promulgated for adoption by the US states which creates a 'sale of goods' styled regime for the licensing of software and other informational transactions. It is argued that the process of transacting software and other informational products is not adequately covered by existing sale of goods type legislation, which finds it hard to classify software. In the case law software is sometimes classified as a good and sometimes as a service; leading commentators to label software the digital chameleon. UCITA aims to avoid this debate by creating a *sui generis* regime governing the formation, performance and termination of information transactions. It has been highly controversial in its content and has only been adopted by two US states: Maryland and Virginia, http://www.nccusl.org/uniformacts-subjectmatter.htm.

[79] *Litchfield v Spielberg* 736 F. 2d. 1352 at 1357 (9th Cir. 1984).

[80] *Tiffany Design Inc v Reno-Tahoe Specialty Inc* 55 F. Supp 2d. 1113 at 1121 (Dist. Nevada 1999).

[81] *Lewis Galoob Toys Inc v Nintendo of Am. Inc* 964 F. 2d 965 (9th Cir. 1992).

[82] *Micro Star v FormGen* 154 F. 3d. 1107 (9th Cir. 1998).

One commentator suggests three factors to consider when establishing whether a later work infringes. First, does the later work "substantially incorporate" or is it "substantially similar" to the pre-existing work? Second, has the copyright holder of the pre-existing work been compensated for such use. Third, what rights did the creator of the later work have to "display or to copy" the underlying work.[83] It is further argued that the copyright owner of the pre-existing work must prove the derivative use "was not customary or reasonably expected" thus denying them opportunity to be compensated for their work.[84] All free and open source developers know it is customary and reasonably expected for other developers to build on their efforts and, often, the only form of compensation in the GPL is the publication of source code for any derivative changes. The harm in misuse of GPL products often lies in lack of specific performance in publishing source code of the derivative changes.

As indicated above, the GPL does not apply to programs "reasonably considered independent and separate works in themselves"[85] which can be distributed within a modified work but not come under GPL obligations. Chief legal representative of the open source licensing certification process, Larry Rosen, argues concerns about GPL's 'viral' nature are exaggerated. "A derivative work is not created by merely touching, any more that one catches AIDS by merely hugging. A more intimate relationship is required."[86] Rosen indicates clear examples where infringement of the GPL work's copyright would not occur. First, a proprietary program that runs a GPL work such as in the GNU/Linux operating system does not alter the GPL licensed work. Secondly, programs that are dynamically linked such as a printer driver for a Linux operating system do not interfere with each other's source code. Thirdly, programs that interact with common data but use an application program interface (API) do not alter the source code of each program. For an infringement of the GPL to occur an author must "consciously recast, transform, or adapt the GPL-licensed software" and then distribute this work under some license other than the GPL. Rosen also objects to the term 'viral'. He sees the GPL license as a bargain between a developer who makes their work 'free' in return for others making their enhancements free. "A derivative work inherits the benefits of the GPL", Rosen claims. He prefers to see the power of the GPL as resulting in "inheritance" of benefits rather than diseased infection.

---

[83] L Loren "The Changing Nature of Derivative Works in the Face of New Technologies" (2000) 4 J. Small & Emerging Bus. L., 57 at 84.
[84] Amy Cohen "When Does a Work Infringe the Derivative Works Right of a Copyright Owner?" (1999) 17 *Cardozo Arts & Ent L. J* 623 at 657.
[85] GPL, Note 40, Clause 2.
[86] Rosen Larry, "The Unreasonable Fear of Infection", *RosenLaw.com*, [http://www.rosenlaw.com/html/GPL.PDF] (23 September 2001).

3.2 Entering the License Contract

At what point is one contractually bound by taking an open source license?

Acceptance of a license is a contractual agreement. To date, case law indicates a licensee accepts the conditions of a software license by opening the shrink-wrap of a program even if the license is not on the box, but inside it.[87] For programs distributed digitally by some electronic distribution mechanism, the licensee must act in a way that plainly manifests assent in a clickwrap agreement. Mere downloading is not sufficient. Assenting action must be unambiguous. "The primary purpose of downloading is to obtain a product, not to assent to an agreement. In contrast, clicking on an icon stating 'I assent' has no meaning or purpose other than to indicate such assent."[88]

The GPL however indicates a different form of acceptance. Assent occurs "by modifying or distributing the Program (or any work based on the Program)".[89]

3.3 International Issues

As the free software distribution model grows throughout the world legal notions such as "jurisdiction" and "choice of law" will highlight the legal nature of the GPL and its international enforceability. Some of these issues may turn on whether we see the GPL as a license or a contract? To date in this article, these words have been used almost interchangeably. The distinction is important. Contract law is subject to the vagaries of various national approaches. For instance, some legal systems require a contract to be in local language for enforceability. A copyright license enables products to come under intellectual property laws that have been harmonised by international treaties such as the *Berne Convention for the*

---

[87] The decision of *ProCD, Inc v Zeidenberg* 86 F.3d 1447 (7th Cir. 1996) held that shrink-wrap, and arguably click-wrap, licences are enforceable in the USA in certain circumstances: *Hotmail Corporation v Van Money Pie Inc* 47 U.S.P.Q. 2d. 1020 (N.D. Cal. 1998); *Register.com v Verio Inc*. 126 F. Supp. 2d 238 (S.D.N.Y. 2001); Lemley, Menell, Merges and Samuelson *Software and Internet Law* (2000) 494-5. See also *Hill v Gateway 2000 Inc* 105 F. 3d 1147 (7th Cir. 1997) *NBA v. Motorola, Inc.* 105 F.3d 841 41 U.S.P.Q.2d (BNA) 1585 (2d Cir. N.Y. 1997); cf. *Wrench LLC v. Taco Bell Corp*.51 F. Supp. 2d 840 (W.D. Mich. 1999), *Klocek v. Gateway, Inc*. 104 F. Supp. 2d 1332 (D. Kan. 2000). Lemley et al suggest that the weight of this decision should not be overstated as it goes against the majority of judicial opinion on the issue: Lemley, Menell, Merges and Samuelson *Software and Internet Law* (2000) pp 490-3; see also B Sookman, *Computer, Internet and Electronic Commerce Law* (1991) Carswell Canada 2-71 ff. Nevertheless the rationale of *ProCD* has been codified in ss 208-9 *Uniform Computer Information Transactions Act* (UCITA).

[88] *Specht v. Netscape Communs. Corp* 2001 US Dist. LEXIS 9073 at [26] per Hellerstein USDJ; 150 F.Supp 2d 585 (S.D.N.Y. 2001) affirmed on appeal, 2002 US App. LEXIS 20714 (2nd Cir. 2002).

[89] GPL, Note 40, Clause 5.

Protection of Literary and Artistic Works, *WIPO Copyright Treaty* (1996) and *TRIPS*. Along with the principle of national treatment this means that copyright law is (arguably) more widespread and uniform than contract.

Chief counsel for the Free Software Foundation, Eben Moglen, suggests the GPL is a copyright license not a contract. [90] "Licenses are not contracts: the work's user is obliged to remain within the bounds of the license not because she voluntarily promised, but because she doesn't have any right to act at all except as the license permits."[91] In the only case to consider enforcement of the GPL requirement to publish source code, he made the following claim:

> The GPL is a very simple form of copyright license, as compared to other current standards in the software industry, because it involves no contractual obligations. Most software licenses begin with the exclusive rights conveyed to authors under copyright law, and then allow others access to the copyrighted work only under additional contractual conditions. The GPL, on the other hand, actually subtracts from the author's usual exclusive rights under copyright law, through the granting of unilateral permissions. When a work of copyrighted software is released under the GPL, all persons everywhere observing its terms are unilaterally permitted all rights to use, copy, and modify the software. Because these permissions are unilaterally given, users who wish only to use the software themselves, making copies for their own use, or who wish only to make derivative works for their own use, do not have to "accept" the license, because they have no reciprocal obligations under it.[92]

An analogy can be made with land to demonstrate Moglen's view. I can give you a license to walk on my land. This requires no counter obligation from you. It remains a unilateral permission not a contract. But such a view ignores the obligation to publish source code on a developer wishing to distribute a derivative work based on a GPL product. This obligation makes the license more like a contract.[93]

---

[90] It could be argued that if the GPL is a mere licence it can be revoked at will leaving developers on a free software platform at the mercy of the licensor: J Malcolm, "Problems in Open Source Licensing" (2003) <http://www.ilaw.com.au/public/licencearticle.html>

[91] Eben Moglen, "Free Software Matters: Enforcing the GPL, I", 12 August 2001, [http://emoglen.law.columbia.edu/publications/lu-12.html] 25 January 2002. See also: B Fitzgerald, "Digital Property: The Ultimate Boundary?" (2001) 7 *Roger Williams University Law Review* 237; B Fitzgerald, "Commodifying and Transacting Informational Products Through Contractual Licences: The Challenge for Informational Constitutionalism" in CEF Rickett and GW Austin (eds), *Intellectual Property and the Common Law World*, Oxford, Hart Pub, 2000, 35.

[92] *Progress Software Corp. v. MySQL AB* 2002 U.S. Dist. LEXIS 5757**.** The "Declaration of Eben Moglen in support of defendant's motion for a preliminary injunction on its counterclaims" made on February 22 2002 is found at: [http://www.fsf.org/press/mysql-affidavit.html] 8 May 2002.

[93] *cf.* J Malcolm, "Problems in Open Source Licensing" (2003). http://www.ilaw.com.au/public/licencearticle.html.

In *Progress Software Corp v MySQL AB,* Progress were accused of intentionally distributing a program called Gemini, allegedly a derivative work of the GPL program MySQL, without source code. The case was an application for a preliminary injunction to prevent Progress and its subsidiaries from sublicensing or distributing the program. The finding of the court was an inconclusive outcome for this test of GPL enforceability. "With respect to the General Public License ('GPL'), MySQL has not demonstrated a substantial likelihood of success on the merits or irreparable harm."[94] In addition, Progress did comply with its source code publication requirements under the GPL during the course of the dispute and this did much to cure the breach according to the judge.

The license or contract categorisation issue is one point of consideration that needs to be resolved before it is known whether the GPL will be subject to the vagaries of local contract law or remains internationally effective as a copyright license. It will be interesting to see how courts respond to this issue. In *Sun Microsystems Inc v Microsoft Corp* the Court explained that:

> Generally a copyright owner who grants a nonexclusive license to use his copyrighted material waives his right to sue the licensee for copyright infringement and can sue only for breach of contract: Graham v James 144 F. 3d 229, 236 (2d. Cir. 1998). If however, a license is limited in scope and the licensee acts outside the scope, the licensor can bring an action for copyright infringement: S.O.S. Inc v Payday Inc 886 F. 2d. 1081, 1087 (9th Cir. 1989); Nimmer on Copyright s 1015 [A] (1999).[95]

4. CONCLUSIONS

Another commentator argues that the differentiation between license as an intellectual property right and contract is not one that creates conflict. Rather they are "two areas of law that have long co-existed and that, at least with respect to one, depend on the other for support in the direction of the goals that are purportedly at the heart of the core legal regime. Copyright and other forms of intellectual property law cannot, and have never been able to, foster active development and distribution of information products in society without relying extensively on contracts".[96] The free and open source licenses use this nexus between copyright and contract to create a new paradigm for creation and distribution of digital property.

The remedying of a breach of the GPL is also an evolving issue. Harm and loss, the traditional paradigms of the atom-minded, are difficult to show in communities where egoboo is the principal form of exchange. Other forms of loss have to be argued. What is most sought from any infringer is an equitable remedy to

---

[94] Note 83, para [2].
[95] 188 F. 3d 1115 at 1121 (9th Circ, 1999).
[96] Raymond Nimmer, "Breaking barriers: The relation between contract and intellectual property law", [http://www.2bguide.com/docs/rncontract-new.html] 1 May 2002.

specifically perform by publishing the source code benefits of their changes to the whole community. In addition, misusers of GPL code would be more wary to impugn if they knew that upon a finding for infringement they could be held to account for profits.

This aim of this chapter has been to overview the legal issues arising from the use of free and open source software. In the technology community the idea of a free and open source software model is not new. For the rest of us, including industry and government, the free and open source revolution has just begun, and we need to learn more……

Chapter 3

Live from Silicon Valley. Views of Free and Open Source Practitioners

LARRY ROSEN
Rosenlaw.com, Silicon Valley USA

DAVID SCHELLHASE
Formerly Linuxcare, San Francisco, USA

YANCY LIND
Lutris Technologies, Santa Cruz USA

BILL LARD
Sun Microsystems, Silicon Valley USA

PREAMBLE

The following is a revised transcript of a seminar that took place on 7 June 2001 at Santa Clara University Law School in California's Silicon Valley. The seminar was convened by Professor Brian Fitzgerald and run as a special session of his course on 'Digital Property', and later as part of the *Intellectual Property in the Digital World Conference* in March 2002.[97] It showcased key legal figures in the free and open source software community and provided invaluable insight into the practical issues facing free and open source software licensing systems in a commercial environment. The seminar was open to members of the Silicon Valley community.

The guest speakers were:

- Larry Rosen, Rosen Law.com and the Open Source Initiative
- David Schellhase, formerly of Linuxcare
- Yancy Lind from Lutris Technologies
- Bill Lard, Sun MicroSystems

[97] Brian Fitzgerald, "Teaching Digital Property", <http://www.scu.edu/law/FacWebPage/Fitzgerald/index.html>, (23 April 2001).

Some of the material that follows was also re-presented by Larry Rosen, David Schellhase and Bill Lard via live video link to the Legal Issues for Free and Open Source Software Conference convened by Professor Fitzgerald at QUT Law School in Brisbane Australia on 3 July 2002.

LARRY ROSEN – A LEGAL VIEW FROM THE OPEN SOURCE COMMUNITY

Lawrence E. (Larry) Rosen is an attorney and founding partner of Rosenlaw.com, a law firm in California. He is a computer specialist. He has extensive experience teaching computer programming and has been a department and product manager in the computer and communications industry. As an attorney, his specialty is technology, but he is also a skilled litigator and negotiator, and a legal advisor to individuals and companies throughout the Bay Area and the world.

He is executive director of Open Source Initiative, a non-profit organization that reviews and approves open source licenses and that manages the "OSI Certified" certification mark for open source software.

Free Software Movement and Open Source

I am not religious about open source. My firm helps clients do both open source and proprietary software licensing. There are many, though, who support free software and open source software with religious fervour. Richard Stallman, for example, the author of the GNU General Public License (GPL), is very religious about free software. He draws distinctions between free software and open source software – I hope I don't offend anyone here – that is like drawing distinctions between the Presbyterians and the Methodists; for someone like me who is Jewish, I often can't tell the difference.

I also think that most people that are out there dealing with software can't tell what the difference is between free software and open source software. But people like Richard Stallman really do draw some very, very sharp distinctions. Here's one way of describing those distinctions. I would call the free software advocates the "socialists" of the software community. (Some people who want to insult them call them Communists,[98] but that is really just histrionics; ludicrous arguments for asserting that open source and free software means the end of capitalism as we know it.) Many free software advocates believe that computer software should be freely available to all as a public right.

---

[98] For a critique of the free and open source distribution model see: Mathias Strasser "A New Paradigm in Intellectual Property Law?: The Case Against Open Sources" (2001) *Stan. Tech. L. Rev.* 4 at para [85] who argues "In reality, however, Stallman's vision suffers from the fact that, as with any communist ideology, its appeal is likely not to be powerful enough to attract sufficient manpower to develop enough free software to make it a feasible alternative to proprietary code".

Free software advocates say that software should be "Free as in speech, not as in beer", but as a practical matter, when using their approved software licenses like the GPL, they really mean free as in beer too. That is because software that is licensed under such licenses will inevitably be distributed, because of market pressure, at zero price.

The word "free" when applied to software puts software companies in fear of their profits, but that fear is unreasonable. As I will describe in a few minutes, even though the price of software itself tends toward zero under free software licenses, there are still ways to make a healthy profit from "free" software.

So open source advocates decided to change the name of the movement. The term "open source" better conveys the "libertarian" notions of open speech (eg, publication of source code") rather than the socialist notions of free software for all at zero price. The leaders of the open source movement created an "Open Source Definition" to identify the required characteristics of licenses that promote the free availability of source code and the free right to create derivative works therefrom.

At least initially, then, the open source movement grew out of a desire by supporters of free software licenses to come up with a less frightening, and more libertarian-sounding, name for the same thing. But fundamental differences, there still are, between these camps.

All free software licenses meet the Open Source Definition ("OSD") and are therefore open source licenses, but the converse is not true: Open source licenses are not necessarily free software licenses. There are many open source licenses that allow licensees to create proprietary derivative works and that don't also demand licensing terms that inevitably drive the price of software itself toward zero.

The free and open source movement seeks to protect **the rights of anyone, anywhere, for any purpose whatsoever, to use, copy, modify and distribute (sell *or* give away) software licensed under a free or open source license.** As a practical matter, **this requires free access to the source code.**

A key distinguishing characteristic of free software licenses, and of some but not all open source licenses, is what I call the "reciprocity" requirement. The GPL, for example, requires that any derivative works that are created based upon a GPL-licensed work must in turn be licensed and distributed under the same GPL license. Under such a reciprocal license, if you create a derivative work of an open source program and distribute it, your derivative work is also open source.

Many open source licenses do not contain reciprocity provisions.

Open Source License Key Criteria

The open source leaders, Eric Raymond and his colleagues on the OSI board of directors, are the libertarians of the movement. They really believe that it's perfectly alright to make money from software. They support the balance struck by Article 1, Section 8 of the United States Constitution, which provides that "The Congress shall have Power ... to promote the Progress of Science and useful Arts, by securing for limited Times to Authors and Inventors the exclusive Right to their respective Writings and Discoveries". That means to them, however, that software copyrights ought to serve public purposes and not just be a vehicle for making lots of money.

The Open Source Definition ("OSD"), which is managed by the non-profit corporation Open Source Initiative, defines certain licensing principles. It is a set of standard criteria for open source licenses. If a license meets those standard criteria, and if the license is approved by OSI, then the "OSI Certified" certification mark can be applied to any software distributed under that license. The certification mark gives people a way of knowing that the software that they're buying is licensed under a license that meets those criteria.

Any license that meets the OSD must provide for free redistribution. That is, someone should be able to take that software and redistribute it for free. While free distribution must be allowed under an OSI-approved license, a license cannot prevent someone for charging for the software, as long as any recipient of a copy retains the right freely to copy and distribute the software, and to use the source code freely to create derivative works. In practice, much software under such licenses will be driven to a zero price.

My 95 year-old mother will do nothing with source code. Having source code available to the average user is of no use whatsoever. But, for other programmers, or for companies that want to take advantage of the source code so that they can fix their own bugs, or when the vendor says, "I'll fix it when I'm ready for it", having the source code is a real advantage.

The availability of source code serves the goals of the US Constitution, in its expression of the purpose of copyright laws in the first place. If you can have the source code available you can learn from it. Programmers can learn from each other. They can figure out what other people have done and they can copy it. It's copyrighted, but you're free to copy. And, because you can copy, you can make better stuff. You can make derivative works. You can make improvements.

Any OSI-approved license must give the licensee permission to make derivative works. It's not just "give me the source code", but "give me the opportunity to do something with the source code, to create new things, to make improvements, and then to do what I want with those improvements".

Some restrictions or requirements on licensee behaviour are compatible with the OSD. For example, a reciprocity provision such as the one I mentioned earlier, says "you can make improvements and you can distribute your improvements but you also have to give them back to me". An open source license requires creators of derivative works to publish the source code.

Any OSI-approved license can provide for the integrity of the author's source code: "If I write code and put it out there, you can't take the code and erase my copyright notice. You can't put your name on it as if you wrote it." It is important that people be given credit for what they do because, all these people out there doing all this stuff for free, what are they getting? They're getting a t-shirt and they're getting their name out there. They're getting some sense of pride. Quite frankly some of them work for less than t-shirts. We have to have a way in the license of ensuring that the person who is contributing to the software is able to protect the integrity of his work.[99]

An OSI-approved license cannot discriminate. You can't say for example, "This software cannot be used for the manufacturing of weapons". You can't say, "It can't be used by the tobacco industry". You can't say, "It can't be used by people of a certain race or colour". You have to make your software available to everyone for any purpose.

In an important sense, what the open source advocates are trying to do is ensure that software licensing promotes their philosophy about software. It is not a matter of trying to force the price of software to zero. It is not a matter of imposing the religion of open source on everybody. Nobody is forced to distribute his software under an open source license *except* when a licensee creates and distributes a derivative work of software that is made available under license with a reciprocity requirement. Even that reciprocity condition satisfies the balance required by the US Constitution, in that it gives the original copyright owner the choice of setting the terms and conditions for the privilege to create derivative works. Open source software distribution relies on the copyright laws just as proprietary software distribution does.

---

[99] In this regard consider the notion of moral rights to attribution and integrity of a work which have long been part of the law of continental Europe. Australia adopted a moral rights regime in December 2000 through the *Copyright Amendment (Moral Rights) Act* 2000. For an overview of the very limited protection of moral rights in the USA see: *Carter v Helmsley-Spear Inc* 71 F. 3d. 77 (2nd Cir. 1995); *Gilliam v ABC Inc* 538 F. 2d. 14 (2nd Cir. 1976).

Derivative Works

It is important for me as an attorney to care about what my clients care about, which is to create good software, to release it to the public, and to get credit for it and to be part of a community. Then they say, they also want to make some money out of it. This creates a tension that I don't think anyone in the open source movement yet knows how to address completely.

One can make money on open source software by selling services. That is not easy to do, and free competition makes high profits difficult to achieve in the service business. Indeed, it isn't just the open source companies that are struggling to make money in this way; proprietary software vendors too have implemented the "support" model and attempt to profit from their software by selling services.

In another example, one of my clients has an open source product that is supposed to integrate or allow the inter-working of instant messaging services. My client makes money by creating and distributing proprietary add-ons to his own open source software. They are licensing their client software as open source and their server software as proprietary.

Another kind of tension going on within the open source movement arises when companies try to get control over what is happening to their open source software. Such companies say, "Take this software, I give it to you under an open source license", but like reluctant game-players, their fingers never leave the ball. They are still latching on. They try to control what kind of derivative works can be created, or whether derivative works can be sold for a profit. Those controls cannot be written into an OSI-approved license.

There are many open source licenses. Which one you use for your software depends critically on your business model. When a client comes to me and says, "What license should I use?" I say, "How the hell do I know! What's your business model? Tell me what you want you want to accomplish. Tell me what you want to get out of your software licensing". Then I may point to a license that already exists, if we're lucky an OSI-approved license. I may copy and modify a license from another company, or write a new one.

Let me just tell you a couple of typical licensing issues in the life of an open source attorney. These issues are things that are interesting to me partly because I don't have really good answers yet.

I have a client who writes proprietary software and his proprietary software can be used with Samba, an open source program distributed under the GPL. My client's software doesn't strictly require Samba, but Samba is one of the programs it links with for certain uses. My client wants to deliver Samba on his CD along with his software. So far that's easy. Even under a license with a reciprocity provision, like

the GPL, it doesn't affect your program if you merely place the GPL-licensed and proprietary software on the same CD.

But there is a slightly more intimate connection between Samba and my client's proprietary program. In order to make his program work with Samba, he had to change Samba. So he took Samba and he wrote some new things in it, some little modifications. And then, he has his program (I'll call it "X" to protect the attorney/client privilege), and he wrote some stuff designed to link between those two programs. The problem is that the GPL says that if you create a derivative work, you have to license your derivative work under the GPL.

Now as a lawyer here's what I told him:

> Based upon the way you described your modifications, you have created a derivative work. You have drawn Samba and your changes to Samba functionality in one box on your system diagram. Therefore, under the GPL, you have to publish the source code of your derivative work, and you have to license your modified Samba program under the GPL.

But what about his proprietary program, X? Since he kept X totally separate from the Samba modifications, I told him "this is not a derivative work of Samba, it's a separate program". I have previously argued that if you make changes to a GPL-licensed program, that is statically linked to your program, then you have created a derivative work. But regardless of the form of linking, if your work is clearly not based, in any way, upon the GPL-licensed program, you have not created a derivative work.

There is as well an open source issue about what is copyrightable subject matter. Suppose you publish a standard, a specification, that tells people how to implement something, and someone reads that specification and goes out and implements it in a program distributed under an open source license. That person sits in a room somewhere and writes code based only upon the published specification. Is what he implements a derivative work of the standard? Is the author of the program subject to license conditions in the license to the specification? Can one have a copyright on a specification?[100] Certainly there is no question that one can have a copyright on a printed version of the specification. Anything that you can put down in writing is copyrighted in that sense. But can you have a copyright on the ideas expressed in the copyrighted work? Just the way I phrased it made the answer obvious. An idea is not subject to copyright. But when an idea is expressed in a copyrighted work, and the work is licensed with restrictions on how the idea can be used, can other expressions of the idea be restricted. That would be incompatible with the philosophy of open source and, in the business of open source licensing, there are interesting questions like that. What is the limit of what you can copyright? What is

[100] *Pacific Gaming Pty Limited v Aristocrat Leisure Industries Pty Limited* [2001] FCA 1636.

the limit of what you can license legitimately? How do you license it? And, who has the right to be the licensor of an open source work?

Let me give you a third interesting example. I have a client who writes a very, very popular open source program and that program appears on computers all over the world. Lots of people use it. It's frequently distributed under the GPL with Linux and other operating systems, and I'm going to call it program Y. My developer client found a hardware system, a piece of hardware with a keyboard and a monitor that is sold by a computer company. The hardware has no disc; instead, inside is a flash card and on that flash card is my client's program.

By integrating my client's GPL-licensed software on a flash card in hardware with other "proprietary software", has the hardware manufacturer created a derivative work that must be licensed under the GPL? That proprietary software links to my clients program in some way. Does that mean that all the other software in that box is a derivative work of my client's software? Or only part of it? What is the implication of hardware being treated as a derivative work of a copyrighted program? Is there a difference between software that comes on a floppy disc or software that comes on a CD or software that is embedded in hardware to the point where it is indistinguishable by consumers from the box itself?

I don't know the answers to these questions! Now that my client's software is on that hardware box, can we force those people who created that hardware to obey the GPL and publish their source code? This question has never (yet) been litigated.

Standing to Sue

Also on that box is Linux. And Linux is also licensed under the GPL, so why don't I just say "Listen client, let's not sue based on your GPL software, let's get the owners of GPL Linux to sue, lets get a company like Red Hat that distributes Linux to sue to protect their rights, and let them enforce the GPL terms!" That's not as easy as it sounds. Who can enforce the GPL? Well, the copyright law is really clear about this. The only one that has the standing to sue in a copyright action in a federal court is the owner of the copyright or the owner of an exclusive right under the copyright law. Who owns the copyright in Linux? The individual owners of each contribution to Linux, each of whom licensed their code under the GPL, can protect their own copyrights, but there is no "big Linux" person with standing to do so. Each contributor owns the copyright to a little piece of Linux, and individually none of them can afford to sue. And a mere distributor of the Linux software, even a company as large as Red Hat, doesn't have standing to protect the copyrights in Linux software, because it merely has a non-exclusive license (the GPL) to copy and distribute Linux.

My client's GPL-licensed program, however, the one that I described above, does have clear copyright ownership. I know because I helped the original author

register the copyright and formally transfer ownership of an exclusive right to a company that thereby has standing to sue to protect the copyright. That's essential for him to have standing to enforce the GPL.

Richard Stallman and the proponents of free software would like as much software as possible to be forced into the free software world. So why don't they expand the reciprocity provisions of the GPL to include "collective works", not just "derivative works"? Referring back to my earlier example, putting Samba together with program X on the same CD would create a collective work.

Obviously, no licensee would accept software that came with a license provision that forced all software placed on the same disk to be open source; such a change would make the GPL unacceptable.

The boundary lines of "derivative works" in the software world are still uncertain. That makes enforcement of the GPL a tricky proposition. What most often happens is that a "cease and desist letter" works to stop activities that breach terms of the GPL. This isn't because of the fear of litigation, but the fear of what the other hackers are going to say. So the GPL licensors win their battles without having to go to court. I think that the GPL and the ambiguity of the GPL has served some people well, despite the desire of lawyers like me to find clear and unambiguous answers to the tricky open source licensing questions I've identified today.

DAVID SCHELLHASE – AN IN-HOUSE LAWYER'S CONCERNS

David Schellhase, works in-house as an attorney with Linuxcare, a Linux services company.[101] He is currently writing a book *Inhouse: The Practise of Law Inside an Emerging Growth Company*. He has also worked as an attorney for a number of law firms in Silicon Valley.

Open and Closed Software Legal Issues

Proprietary

I want to contrast the worries I have as a lawyer, being from a proprietary software company as compared to an open source company. Lawyers worry a lot about a lot of different things. We let business people worry about making money. We're worried about keeping money. In a proprietary software company, you have far fewer worries – it's a closed system in effect. You worry about employees, you worry about customers and you might be worried about some people whose products are infringing. If you're at the centre of this system, and it's a relatively closed system, it's a very closed loop in effect. I'm worried about getting rights from my employees, I'm worried about giving the proper rights that my customers

[101] www.linuxcare.com.

deserve in a license agreement and no more. Basically I understand my universe very well. All these employees have signed a proprietary information agreement or they don't come to work for us. All the customers are going to sign a license agreement that limits our liability to within acceptable limitations, which gives us some recourse against them if they're out giving our technology away for free and so forth.

Open Source

In an open source company there's lots more to worry about. From the open source company view I've got not only my employees and not only my customers, but there are dozens, maybe hundreds, maybe thousands of potential owners of intellectual property who are really outside this neat little closed system. Plus, there are hackers who might be contributing to some of my ongoing projects for customers. I'll just give you one example of technology that the company that I represent has worked with a lot. It's a technology called Samba that is a file print sharing technology for Linux. It basically turns Linux into Windows NT in effect. Samba was written by a former employee of Linuxcare, but long before he got to the company, so it exists somewhere out there in the ether on samba.org. Many customers, specifically hardware OEM's (Editor's Note: Original Equipment Manufacturers) who are interested in proliferating their hardware devices, are interested in Samba because it seems like a cheap alternative to Windows NT. In combination with Linux it seems like a free alternative. But they need to optimise their hardware boxes, so they call Linuxcare and look for us to consult. When that happens I've got a huge number of worries. I'm worried about all kinds of potential owners of property. I don't know what's in Samba. Samba is a million lines of code. I don't know where it came from. It pre-dates my company by many years.

The Samba organisation hasn't necessarily said we're going to give you all the rights that we have from our contributions to the Samba code. And if we're making new contributions to the Samba code, we may want to give those back to samba.org. We may not want to give them to you, so you can't give them to your customers. So your customers who are paying good money for the delivery of some kind of code, maybe won't get the indemnities, limitations, liabilities and warranties they expect from a proprietary software vendor. That's a big problem, getting big companies, big hardware OEM's over the hurdle of understanding that they may not get all the nice warm fuzzies.

[The following table, which was handed out by David Schellhase to the seminar participants, highlights these points.]

TABLE 1. SELECT LEGAL ISSUES IN SOFTWARE

Open Source Products			**Proprietary Products**	
Concern level	**Ability to do something about the concern**	**Legal issue**	**Concern level**	**Ability to do something about the concern**
(Pre-release)				
High	Medium	Employees holding back rights	High	High
High	Low	Copyright and trade secret infringement	High	Medium
Medium	Very low	Patent infringement	High	Medium
(Post-release)				
None	Very low	Warranty	High	High
None	Very low	Intellectual property infringement indemnification	Medium	High
None	High	Unusual license provisions (channel readiness)	High	Medium

Employee Problems

Our biggest problem is with the employee base and hackers. The employees don't want to sign proprietary information agreements. They don't want to say, "You, the company, own every piece of our work product". That's not what they're interested in. If they were interested in money, they probably would have developed Samba's proprietary program and would have sold it under a more proprietary licensing scheme. It wouldn't be GPL'ed. It wouldn't be out there for free. So they're motivated by different things. It's very easy to motivate people who have bought into the capitalist system. It's very difficult to motivate employees who have a different incentive for coming to work everyday, because we're just not sure what that is. Corporations in the American capitalist system aren't set up to reward people with warm fuzzies or whatever else it is that people work for. So, I have got a lot of worries about my employees.

I've got to figure out a way to get what Linuxcare needs and to deliver to customers what customers are demanding out of these employees who are out of an organisation that exists in the ether and isn't beholden to anybody. My employees may work for that organisation and contribute to it from time to time, but they're not identified as Samba Inc, it's Samba.Org. They operate by consensus largely

and many of those people involved in that organisation don't work for me, they may not like me.

Hackers

Furthermore, I'm worried about these hackers because Samba is an ongoing project, and there are a lot of hackers out there who are contributing to it. When my employees are on a job for a customer, they may use some of the code developed by these hackers that isn't yet in the mainline Samba code. They may be using bits and pieces of new stuff and I don't know where they got it. They may be asking their friends, half a world away or more, to write a few lines of code and give it back to them by email overnight. This happens all the time in this world. That's a very difficult thing to understand coming from a proprietary world. It's also very difficult for customers to understand. So the number of worries goes up astronomically, maybe exponentially, as you move from the relatively closed system of the universe that you understand, to the much more open system of the universe that you don't understand and may not know about. Lawyers obviously fear the unknown, because they can't control the unknown. There's a lot more fear and a lot more worry here. The crux of the matter, the relationships with the employees, is critical. Convincing the customer that they are not going to get all of the rights that are used to getting is the other big hurdle. So you've got these two big hurdles and lots of ancillary worries along with it. In some respects it's a daunting challenge and it's interesting that Bill said it costs more to give your code away. That may very well be. I don't know of a truly profitable open source company yet and it's unclear whether there will be one. Linuxcare has raised close to 80 million dollars in venture capital. We're down to less than 10 million and it's unclear that expenditure wasn't just handing our customers a bunch of very steeply discounted consultancy services. That's not clear, we don't know.

I'm really more worried about customers. I'm more worried about HP than I am about a hacker in Czechoslovakia. A customer might sue because a piece of code, which we have said is good code, which we have good rights to, wasn't written by us. So a hacker comes along and says, "Oh, I see that HP is incorporating a line of my code. I'm going to go and sue Linuxcare, or I'm going to go and sue HP". HP is going to say, "well these guys indemnified me, these guys said they created the code". Even if there are no warranties and indemnities in the licenses, try convincing a big multinational hardware company that they don't deserve the indemnity. Sometimes you just hold your nose and hope.

A more likely scenario is one of our employees found a piece of code and said this is out there under the GPL, or under some other license, and it's an elegant solution to my little problem which has been vexing me all night, I'm going to take it. A consulting company model like ours gives the rights in what we are doing for the specific customer. You give them the ownership of it. You may retain a license back but typically you give the rights to them, so they feel that they have ownership including copyright and the ability to patent it. So you need to make

sure that your getting a full license back. They may actually own that code. This has not been litigated. A number of actions have been settled where, under GPL in particular, a code found its way into a product, and was discovered. Typically what happens is they take the code out or publish their sources and it goes both ways.

I think what you're seeing in the open source world is some very strange bedfellows. And it's not clear to me that the marriages are going to last. I think there are some inherent problems between the people who believe in community values versus the sorts of folks who want to use open source software for commercial purposes. I think that there could be a big dust-up, and the dust-up is coming. It may not be a big dust-up, it may be that the two go their separate ways. I think they have had a flirtation, they may have gotten engaged, they may now actually be married, but I don't think it's going to be too long before there is a separation and maybe even a divorce. The people who are most interested in open source software are historically the larger corporate hardware vendors. They are interested in ubiquity for their hardware platform, and, for want of a better term, they are software agnostic or operating system agnostic. They love Linux, they love Samba, they love some of the other open source software when it fits their purposes. When it doesn't, or if it turns out it doesn't, there will be hell to pay, or they will go back to the closed system that they're used to.

Don't get me wrong. There is plenty of risk in both kinds of models as companies are finding out. These other owners of software, people like IBM, have hit up companies, famous companies that you all know the names of, for millions and millions of dollars in patent licensing revenues over the years, and will continue to do so. The whole landscape is fraught with peril, but people continue to do business. It's not clear to me that the hacker community or the open source community is going to want to continue to do business with the larger hardware vendors when they figure out what the hidden agenda is. And the hidden agenda is that monogamy is great, as long as you know it is with me, and my systems. You know: "Don't cheat on me with HP or IBM or Compaq or anybody else. Stick with me".

I got a call from a friend of mine the other day who is the CEO at another open source company of fifty people or something and he said: "My open source developers are in revolt". He meant that it is really difficult, because these people tend to march to their own drum. So the answer is, you do it very delicately and I will give you some prime examples. We have gone from 280 employees to 30 by the way. When we had 280 employees, not everyone of them had signed our proprietary information agreement. That's a big problem, and that necessitated some interesting dances with our customers. We have had these issues where we have had to go to customers and say: "You know what, we don't have all the rights you want. You are going to have to sign a license that doesn't look anything like your Oracle or Microsoft license. It basically says: 'You're going to take a lot of risks with us, and we've all got our fingers crossed. We think that these things are OK and we don't think we are infringing, but we can't give you ownership rights

or a good warranty'". That all went back to our handling of relations with employees. We didn't want to alienate them by forcing them to sign what they viewed as an onerous, burdensome proprietary information agreement that everybody in Silicon Valley sort of signs when they join a company, and forgets about. So the answer is I think people are still confronting it.

Luckily we are a consulting company. We don't have the problem some of the product guys have, which is: Why should I pay you for something that is available on the web free? That is the Red Hat problem. We don't have that problem because we are providing bodies and bodies do have a cost. For a senior developer we can still charge $200 an hour. Even though you may not own my product you still get some kind of a license to it, or you get to pick that person's brain and maybe get your people to code it or something like that. So it's not a perfect model and it's really not perfect if you're a big hardware vendor used to squashing everyone and getting your way.

Patents

I'm less worried about patent infringements. The potential patent owners, the open source developers, are also the ones typically interested in spreading and proliferating open source technology. If we're talking about a hardware vendor or somebody who is a software vendor, whose application sits on top of Linux or something like that. These guys are much less worried about the hackers because they tend to not have the resources that leads you to worry about patent litigation. I mean that's a game for rich people. These people tend not to be rich. And they tend to be free about their intellectual property and they're less likely to bring the patent suits.

If I've got huge economic resources, if I'm big enough, they will come calling on me. We will do some kind of cross license because I'll have some patents too that they're probably infringing. So I agree in the abstract, it is a huge concern, and it can paralyse you, if you really think hard about it, because there are so many elements to it. But, I'm not so worried about it on a practical day-to-day basis. Worry about the other stuff.

Being a defendant in a patent infringement case, particularly in software, is a high class problem to have. I'd love to have that problem, because it would probably mean I would have a hundred million more in revenues. It means I'm probably with a profitable and successful company.

YANCY LIND – A BUSINESSPERSON'S VIEW

Yancy is CEO of Lutris Technologies in Santa Cruz, an Internet middleware software company.[102] He is a businessman, not an attorney, and aims to make open source companies commercially successful.

Lutris Technologies and Java Application Server

We are in a really interesting transition phase right now where it's not clear at all if the marriage between corporate America and open source is going to survive. We make this thing called an open source Java application server (JAS). JAS is an important piece of middleware software. If you're going on the Internet today, you typically don't go to web sites for very long. Apache is out there, but you quickly get handed off from a web server to a thing called an application server. An application server is what makes the Internet run today, it's how you get to back-end processes like databases and e-commerce systems. The JAS is the critical battleground right now in the software community. Whoever controls the JAS will control software computing for the next 20 years. That is why in a list of 5 types of licenses put up there by Sun, there is one license that was exceptionally closed compared to the others and that was around Java.[103] Because Sun realises that the most important piece of technology that they have today is something called Java, and it's got their most restrictive license around it.

So, what is open source software? The answer is – it is wide open for debate right now. A lot of companies and groups have shared source code and software development over a lot of years and it really has had some wonderful things come out of it. You know TCP/IP is a great example of the early days of this whole idea. The term open source software came out of the Freesoftware Foundation because they realised you could not go around selling free software and have it accepted in corporate America. So they had to come up with a new name and called it open source instead.

There's really good reasons beyond that. There's this idea in the open source community that we want to have this freedom of intellectual purity. We want to be able to share ideas, but we don't necessarily want to be able to rip each other off. So there is this very famous saying in open source that says, "Open source is about free speech, not about free beer". And that means we're all about sharing ideas, and working together to make things better, we're not all about giving away money. We still have to figure out how to make money.

We don't think the GPL is a good license frankly, because of the viral aspect of it. The core issue is "Are you interested in satisfying the hacker community or are you

[102] http://www.lutris.com/.
[103] See the section of this chapter on the Sun Community Source License.

interested in somehow making money?" That's really what it comes down to in a corporate sense, not as an individual. The GPL in particular applies to this ethic. This whole idea about the hacker community and people who are just individuals out there making sure that they can contribute and that they can be in this large group of alpha geeks. My code is better than yours. I can do that better. There is this whole ethic around that and the GPL is a license that appeals to that ethic.

What is it about open source that really appeals to me as a businessperson? Well, it's this idea of innovation. It's just a wonderfully powerful idea. We really do have this large three thousand-member development committee who actively work with us and innovate and share ideas with us. We've gone off in directions and had wonderful breakthroughs because people were working with us. The lack of a single vendor dependency is just an incredible benefit. Self-reliance to fix bugs – companies do like to have the source code. They like to know that their alpha geeks on their staff can get inside that product and do something with it. One of the really big issues in software development is relying on the vendor's release cycle. What if I have a bug that has to be fixed now and I can't wait for three months for your next release cycle.

Case Studies – Plantronics and General Electric

We go into big companies and build things with them. One was called Plantronics. They are the world leader in headsets. They had us come in and start building a very large e-commerce system for them. They started out using [a major corporation's product], which is a direct competitor to my product. They started finding some bugs in the product and it didn't do as advertised. They went through all the traditional kind of mechanisms that the competitor provided such as support groups. Finally there is just a bug and how does it get fixed? They call up the competitor.

> "Please fix this bug".
> "Well, how many copies are you going to buy?"
> "Well, we're going to buy one."
> "Okay, we'll try to get to that bug in our next release."
> "When is that going to be?"
> "Well we're not sure".

And there they are, dead in the water.

So they threw out the competitor's software and Lutris came to the rescue. We were able to show them the power of having access to the source code. Use our application server versus that application server and if you have a bug in it we will fix it for you right there, or you can fix it. Whatever the case, there is the code. Here's all the build utilities, all the make files, all this kind of stuff – just go!

I'll give you another example. GE is one of the largest companies in terms of market capital in the world today. Their largest division is the home appliance division. They make things like washing machines. They recently threw out [a rival company's] application server in favour of my product. They did that for a very similar reason as Plantronix. One of the great things about Java is that it can be decompiled. [Editor's Note: a decompiler takes executable code of a program and turns it into source code]. That means that people like Sun don't like that, but developers love it, because you can still get access to the source code. So what happened was GE found a bug in the rival's application server. They wouldn't fix it, wouldn't give them any time of day. So internal IT guys at GE decompiled the application server, found the bug and sent the fix back to our rival company. They said "How neat, we'll get that into our next patch release!" And, GE said "That's great" and ripped it up and put our stuff in.

Two very powerful real world examples of why open source software is a wonderful, wonderful tool. True value to customers. Value you could never get from a closed source product. From a businessman's perspective I have got a product that competes with the giants of this industry who have hundreds of engineers working on these products. I have forty. My product many consider to be superior with a fraction of the engineering costs. So, just from a pure return-on-investment perspective, open source is something I can leverage massively to give me a real economic advantage.

Commercial Viability of Open Source

So, open source is a great thing. Is it working? The answer is yes and no. Unfortunately there is a no to it. It was only three years ago that open source software really came on the scene from a mass consciousness kind of perspective. They're talking about it because of the fact that there is a wonderful PR angle. There is this David versus Goliath angle going on with Red Hat versus Microsoft. Three years ago Red Hat had not gone public yet and there was this thing called Linux. And there was this great article out there called *The Cathedral and the Bazaar*.[104] Press people were drooling over this stuff. And suddenly open source software burst on the consciousness.

A lot has happened in the last three years. Open source software has moved out of the fringes and is the main stream today. There is no doubt about it. Open source software has absolutely proven itself as an extremely powerful software development methodology. The best application server in the world is mine. And it's open source software. The best web server in the world is Apache, there's no doubt about it. It's open source software. Many people would consider Linux to be

[104] Eric Raymond, *The Cathedral and the Bazaar*, version 2, 24 August 2000, <http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/index.html> (27 July 2001).

the best operating system in certain niches. It's open source software (OSS). It's clear that OSS has proven itself as a software development methodology.

But it's not at all clear if there will ever be a successful stand alone software company using open source. Because, the only way it looks like you can make money off OSS is by packaging it with something else. It's a great way of maintaining margins for existing products. Hardware companies love OSS because they can maintain price. The number one problem that hardware companies have today is maintaining margins. That's it! And you can only manufacture your margins for so long before you just get to a certain point where you can no longer squeeze another penny out of your manufacturer. So you have to bundle stuff in and open source software is great for that.

So, some examples as to why hardware companies love this, or some proof of this. IBM this year is putting one billion dollars into OSS. A lot of money right? But for them, not that much. It's a way to harvest out of the OS community, package that stuff into their hardware, and be able to maintain their pricing of their hardware, maintain their margins. Hewlett Packard have stated publicly that Linux will be their only operating system within five years. HP-UX will be gone. They won't be selling any Microsoft products. It's an amazing statement, unbelievable. Compaq right now is the number one Linux platform in the world. And even Sun is making some moves towards OSS. Everyone is moving there.

But OSS has not created any viable stand-alone software companies. There is not a single example of success. Lots of failed struggling companies. Anyone in this space right now is hurting big. And it's not clear if it will ever be profitable. And at the end of the day, if it completely collapses, we're going to be back to the good old business model that I used to have to worry about which is profitability. There's not a single OSS company that's anywhere close to being profitable.

It's an interesting time to watch what's going on here, because what we are seeing is these huge computer companies saying: "OS is the future", but software companies struggling with how we're going to make it. My company included. And, the other interesting part is if you look at most of the OSS projects out there today – they're not alpha geeks – they are actually employees of big companies, making it happen. Most of the developers who work on Apache today work for either Sun or IBM. That's just the truth, most of the developers who work on Apache today are doing it because a large hardware company has an economic interest in seeing Apache succeed.

BILL LARD, SENIOR DIRECTOR OF LICENSING STRATEGY AND ARCHITECTURE AT SUN MICROSYSTEMS

Bill Lard is Senior Director of Licensing Strategy & Architecture at Sun Microsystems, Inc. He has been an Attorney with Sun for nine years handling software related matters. His current role is to establish the future direction of Sun's overall technology licensing strategy and architecture.

Introduction

This section of the transcript looks at a Sun public license that does not have OSI approval and is called a "community" license – Sun Community Source License (SCSL) Version 2.3. This is not to be confused with the Sun license that does have OSI approval as shown in section 2.5 of this article. Sun has slightly different SCSL's for each product. The license considered for this article relates to their Java 2 Platform that provides "Write once run anywhere" capability for applications developers.[105]

The main features of the license are that it:

1. requires developers to become a community member by entering into the license;
2. under research use rights, limits distribution of source code of the original contributor to other community members;
3. requires licensing back to Sun of source code for 'Error Corrections' "as soon as practicable";
4. allows distribution of fully compatible, object code to third parties as part of a value-added product under a license of their choice, consistent with the SCSL once Sun and the licensee have signed a "commercial use" attachment; and
5. compatibility of licensee implementations must be determined by use of the Technology Compatibility Kit supplied by Sun.

Serial Licenses

An important consideration not really legally tested today is that true open source licenses are entered into serially. You can receive source code in a chain that was originated by someone 100 people before you, so privity of contract starts to wither away. Now the GPL supporters believe that anyone who makes a contribution to an open source bundle who has copyrights, and has retained those copyrights, also retains the right to sue on copyright infringement based on the copyrighted code in that bundle. So if you download some code that's got 1000 contributors and you

[105] Java 2 Software Development Kit version 1.3.1 Sun Community Source License <http://www.sun.com/software/communitysource/java2/index.html>, (23 February 1999).

utilise it in a way that's inconsistent with the license, you've got a potential claim by any one of 1000 people for copyright infringement and possibly breach of contract. But like I say, it's not tested, so we don't know whether the courts would find standing for an early contributor whose contribution had been substantially diluted over time.

Viral Nature and Inheritance

The Linux operating system is available under a combination of GPL and LGPL and the distinction here is really important. They require you to license your technology back, under the very same license, not just any old license. You may have heard of the viral impact of GPL code. The Free Software Foundation prefers to use another term, "inheritance". To put this in context, Sun has a proprietary operating system called Solaris. It's our Unix environment that we use for all of our products and everything we ship is based on it. It comes from a very long heritage of technology that was developed in Berkley and AT&T and then after a while it was licensed under a proprietary license by Unix Systems Laboratories, a subsidiary of AT&T. Over the years the Solaris code base has acquired a substantial amount of third party technology that had confidentiality requirements and a variety of other restrictions. That technology cannot be, in whole, made available under an open source license because of this contractual baggage. If we were to take GPL code and integrate that with Solaris, there are circumstances where we could wind up with an obligation to open source the Solaris code base under the GPL. It could create a situation where we're either in breach of the GPL or in breach of contract with a bunch of folks that had contributed to the Solaris code base over many years. So this is a really critical concern if you're going to use technology that's licensed under either GPL or LGPL.

I might mention that the LGPL code was designed to provide for interaction with proprietary software, but primarily for libraries. It was designed to do that where the free software community wanted to make sure that their libraries were used because there was more benefit to having even the proprietary guys use them, and make them standard, than to hold back and try to force people to have their code become open. That's an important distinction and it's also being used now in a way that I think is very interesting. If you want to take GPL code and use it in some way in conjunction with proprietary code, you may be able to do so if you use LGPL code as an abstraction layer between your proprietary code and the GPL code. It's important that the GPL code is dependent on the LGPL code and not the other way around. Otherwise, the LGPL code may be converted to GPL code and your proprietary code may be affected as well.

Sun Community Source License: A Non-OSI Approved 'Public' License

So here's the one the open community guys have given us a hard time about - The Sun Community Source License. It is a public source license, but not an open source license. It provides technologies that were developed at Sun in conjunction with many of our industry friends. It makes the code available publicly, but under a different licensing model. The reason the community license is used is because the number one value proposition behind Java technology is compatibility, "Write Once, Run Anywhere". You can write an application once, and it will run on any Java platform regardless of the microprocessor and operating system if it's a compatible implementation of Java. In order for that to work we need to make sure that people do not fork the code[106] and create non-compatible implementations and take it off in a different direction. That being said, in the industry, even in the open source world, people tend to manage compatibility very well. If you look at Linux, there are six flavours of Linux, but they're not that different in flavour. The difference here is that the community developing Linux has an incentive to have a compatible set of implementations of the Linux operating systems so all Linux applications will run on any of them.

In the case of Java, there is at least one company that would prefer to see one platform rather than applications that run on all platforms. They have the ability to basically take the developer base and move them away from what would otherwise be applications development for a broad set of microprocessor and operating system platforms. You can't just say we'll trust the world to make sure this doesn't happen. So as a result of that, we do a few specific things. One is, we have a public source license. So if you want to get Java technology you have to go to the website, click on the license and accept it, and that puts you in privity of contract directly with Sun. That takes care of many concerns about enforceability and standing.

There are a couple of other things that are important about the community license. It's not fully open to the world. We only allow downloads to countries where we are comfortable that intellectual property protection and enforcement is reasonable. There are about 50 countries on the list. The rest of them currently are not available to download. Granted, you can take that technology, download it to the UK and get in your car and drive to Libya, well with difficulty, but I mean, you could do that, so there are those that would argue that we are not protecting much. Nonetheless, we think it's important to limit distribution to those areas where we know Sun will be able to enforce the license requirements. So that's basically the Community License.

---

[106] Code forking involves creating a derivative or modified, privately controlled product that has not complied with industry standards and thus leads to a multiplicity of forked private versions. See: Marcus Maher, "Open Source Software: The Success of an Alternative Intellectual Property Incentive Paradigm" (2000) 10 Fordham I. P., Media & Ent. L. J. 619 at 678.

Issues in Public Licenses

So why a public license? I mean what in the world would possess someone to give away technology? In the 70s where developers at universities were basically using their operating systems as their research base, if someone working on code had a problem with it, finds a bug, or it locks up, if they leave for the day and are the only one that has access to the source code, the guy that comes in and wants to work in the evening cannot use the system because it has got a problem. If you make the source code available to all the researchers that are running on that server, and someone has a problem during the evening, they can go in and fix the bug, if they're competent to do so. So this evolved into group development in a relatively closed way, that grew to be much more worldwide over time, particularly with the Internet. Starting in '92 with Linux in particular, but from '95 on, when the worldwide web became the vehicle for getting access to the Internet, the amount of people exchanging code in development just exploded worldwide. That's why you've got so many thousands of developers today that are working with Linux. So, encouraging community contribution is one of the key things, and the idea there is that innovation always happens elsewhere. If you only rely on your 10 or 15 or 100 employee engineers to develop technology, you're not going to get the benefit of the other thousands that are out there that might be willing to participate in your program if the circumstances are appropriate. What are appropriate circumstances? Have you set up the right cultural environment for the community? Have you provided the right incentives for developers to participate and get benefit from actually developing in that community?

Standards are another really important area. If you want to get technology out there and have it adopted as a standard, having the source code available to people so they can get access to it and use it, is a very good way to go. Again, you still need to be able to interact with the community and invest the time and money to make that community work; otherwise, it is not likely to happen. We've got a number of projects that are out there today, in file sharing, and transfer of files over the Internet. We want very much to make sure they stay open standards – and we're willing to make the code available to assist in that happening.

Generating revenue – now this is primarily with the community license, because we reserve the right to charge people royalties for distributing products that integrate our code. That is not something that you see with open source in terms of distribution of the source code itself. The open source guys can charge for distribution of binaries, but they primarily look to other means to generate revenue. For example, the Red Hats of the world provide both support and professional services intended to generate revenue. Whether that model is scalable remains to be seen.

Capturing developer mind share is another important issue. Sun has a number of platforms where we want to encourage developers to write applications. Having

access to source code is very helpful to be able to debug programs and to write better applications, because if you can actually see how the operating system works, your applications can take advantage of that. So, we have a number of programs to make technology available for that purpose.

Also, Governments – actually there is a lot of discussion right now about whether various Governments are going to require that only open source be available. The Government of Denmark has actually been looking at that. They've got Microsoft terrified right now because they're suggesting that maybe they should just only use open source technologies for their operating systems, rather than the closed environment that Microsoft sells. Well, not a great idea if you think about it. If you only allow people with open source to sell to the Government, you are going to have most of the proprietary systems that are around here today, not being used by the Government. I mean, certainly we couldn't do it, because Solaris is not open, can't be, at least not today. Microsoft could not do it. Same thing with HP and IBM – although I imagine some of their systems can be delivered on Linux. But, its not always a bad idea, Governments often require access to source, usually in escrow. Sometimes it's easier to just do an open source arrangement for them and make sure they have access to it that way.

Goodwill is another possible goal- simply making the code available because you are not using it. At Sun, and many companies like us, we have lots of projects. Sometimes we have projects that are focusing on very similar things at the same time and one will win and one won't. So the one that wins goes off and becomes a product someday. The people that worked on the one that didn't are thinking, "I put a few years into this thing and look what's happened – maybe we should give it to the world". That might be the right thing to do in some circumstances, but there are costs associated with doing that. There is a lot of hidden costs associated with properly handing code off publicly, so you really have to weigh the benefit of making it available for free, versus the costs associated with it. There is a lot of code scrubbing that has to be done to be sure that it is suitable for public consumption.

Posting Code

So what about the actual process of posting source code publicly? Once you make the decision that you want to do it, how is it actually done? What should we be concerned about?

From a public perspective, make sure the code is of reasonable quality and useful to your target audience. You don't want to throw your garbage in the street as it were; it's considered to be bad form. For the most part, if you just have a project that died and you just want to get the code out there, to say "Gee what a nice thing I did", it probably won't work. You also have to make sure there aren't any inappropriate comments in the code, because when a programmer's up in the middle of the night, irritated about something, casting aspersions on Bill Gates in

code comments is not the thing you want to see out there the next day. So you go and do research, do scripts that look for key words or peoples' names, foul language, what have you. You can also do a script to search for third party copyright notices. If we didn't realise we had some code from a third party, making it publicly available suddenly makes it known that you did, and you have potential liability. So it is good to make sure that the least you have done is a reasonable search for third party stuff.

Programmers may or may not realise they have developed patentable inventions in their code. Also, the code may read on patents you already have filed and have issued that are sitting in your patent portfolio. So we need to take a look, check with the patent database and look for anything that might be affected by publishing the code. If anything is patentable, you have to decide if you want to file on it before making it available to the public.

If you intend to encourage community development, you have to provide incentive for people who are not into monetary rewards to hack the code and post their contributions. It usually has more to with things like the personal satisfaction of having provided a really cool piece of code and see others actually use it.

So you got the code out there, its clean, looks good, you have got a community working and people are actually providing contributions back. What do you do about that? If you went out under a Berkley style of license then if people are providing code back there's not a mechanism to dictate the license terms under which you receive it. For example, the Apache foundation using a BSD style of license also requires a contributor agreement for major contributions to their code base. So if you want to provide code, you need to sign an agreement that says, "I own the copyright, I wrote this myself, my employer does not have copyright under my employee agreement, and if I know of any encumbrances in terms of other intellectual property that might require a license, I will tell you so that you can decide whether to take it or not". This is a very key concern because if you allow code to come in and you do not know where it came from you can't be certain if the full rights are there.

Posting under GPL

If you put code out under GPL, someone can take that code and make modifications to it but it triggers an obligation to make their modifications available under the same license. If you want to create a community under GPL where people are going to provide code back, then it all has to be GPL code. So if you have in mind to do something else in addition to GPL then you need to have ownership in that technology because the owner of the copyrighted code has the right to do what they wish with it. They still have the obligation to license under GPL if the code was derived from GPL code, but they have the right to license it under other license terms as well. Hackers often license the same technology under a GPL and a proprietary license so they can charge for the proprietary version. If

you have GPL stuff out there people are not too excited about going and getting it, especially commercial entities, so you can say, "By the way, I can license you a propriety version as well and it will only be $100,000, great!" That's one way open source developers make their money.

Employees

We're actually working on a program that would allow for variations on our employee agreements because we have probably fifty employees that are contributing to Mozilla and other projects. We have a ton of other employees doing Linux work and a variety of other things including Gnome.org. Those are all projects sponsored by Sun in some way or another. We can sanction that, but we also encourage our employees as individuals to participate in research programs, because we think it's a good thing to do. The difficulty is if I am in the server group and I do web servers for Sun-iPlanet and I'm busy throwing stuff over the wall to Apache, that's probably a problem because if you read the employment agreement even with the California labour code requirements there's a conflict. So what we're looking at is providing in advance a waiver to that requirement for employees who want to participate on a particular program. It would be for whatever period of time that they want to do it. But it wouldn't be for any open source project there is. What we are grappling with is how to weave this process into Sun's conflict of interest policy.

Downloads

So downloads. This is where you bring open source code back in house. What are the issues? Where did it come from? Did the people that contributed knowingly or not knowingly provide code that was infringing others' rights? That's a hard thing to know. You can do a contributor agreement and get some kind of commitment that they had a right to license it.

Looking at who created the technology itself is helpful. If it is well known developers who have been working in the community for a long time, it is likely there would be a reasonable level of comfort that they have the right to provide the code. Pride of authorship is important to these developers. If the product that you actually put the technology into is critical to your business, and you are unable to ship it due to an injunction, then you think twice about what technology goes in and who wrote it. You don't want to compound the problem by having the potential for copyright claims that come in from code that was improperly provided. I have seen through our own processes where a list of copyrights and a license associated with a specific technology turned out not to cover all of the included code. On further examination we found references to the Free Software Foundation, which means GPL licensing terms. Under the circumstances, we couldn't use the code as planned. So a very careful examination of the technology is really critical.

Conclusions

What this all means to Sun, as a corporation is that there is a place for public licensing. No question about it, we use it and encourage it, we support a lot of programs, and we get a lot of benefit from it. You have to choose your license wisely, whether you're licensing out, or bringing technology in. You need to weigh your liabilities. People think that it is cheaper to go and get code off the web and use it in their products. Sometimes that's not so because of your attached liabilities. Also creating an open source development program to have the community develop the code for you may seem a lot less expensive, but often it costs more to do that, than it does to develop it yourself. You need to have other reasons to do the open source, so that you get the benefit of going out there and bringing in technology.

Chapter 4

Open Source Software: An Australian Perspective

PETER CJ JAMES
Partner, Allens Arthur Robinson

1. OVERVIEW

Open source licensing can result in enormous savings in programming costs, shorten development time and assist in identifying and fixing security issues. However, the success of an open source project depends on capturing the loyalty of the programming community, which may require that particular forms of open source licences are used – even when those licence terms do not work from a commercial perspective.

That has led some open source projects (most recently, the Mono Project to clone an open source version of Microsoft's .NET development platform), to adopt a level of commercial pragmatism – allowing large software companies and embedded system manufacturers to develop their own closed-source derivatives of the software to secure those companies' involvement.

That is at odds with the ideal of the "copyleft" purists (who say that all software which is derived from open source software should also be made available to all on an open source basis), but is good news for the commercial future of open source.

The commonly used GNU licensing model does not adequately deal with issues arising under Australian law. The self-replicating nature of the GNU model means that the opportunity to redraft the licence to rectify these problems is constrained – except by expanding the licence terms with additional exclusions and limitations of liability. This leaves the supply chain for products which include GNU-licensed software potentially exposed with liability under implied warranties (discussed in section 5.2), liability in negligence (see section 6.2) and liability for loss of profits and loss of data (see section 6.3).

This paper examines some of the practical issues involved in open source licensing in Australia and what the ideals of open source have in common with business reality in this country. With most developments in this field originating in the USA, this paper also examines some of the difficulties under Australian law which arise from using commonly available open source licenses.

Watch your language

Before looking at the particulars of open source licensing – a cautionary note: Whether advising a developer, distributor, user or service provider, a lawyer's enemy (or best friend, depending on which side in a dispute is being taken) is imprecision in the expressions used in the industry. Words have a range of, often conflicting, meanings depending on their use.

Included in Schedule 2 is a selected glossary of expressions used in open source discussions and literature. Be careful with expressions like these, since industry usage of the expressions varies greatly. Defining exactly what you mean in a licence, website or software documentation is the best approach, to avoid any ambiguity about usage of a word.

One example is "Freeware", which many use to describe the freedom of rights of use and distribution attaching to particular software, but not necessarily implying the use is for no monetary charge – "free as in speech, not free beer".[107] Used this way however, "Freeware" might still have conditions of use attached to it (for example, an obligation to report and share improvements or to distribute the software and derivative works on the same licence terms as the original code) and therefore be something more restrictive than "public domain software". However, others use "Freeware" to describe no-charge software which, again, might not necessarily mean that there are no licence restrictions applying to it, despite the rights of use being for no charge.

Using industry jargon does not necessarily convey a precise meaning and can lead to misuse of the software or dilution of legal rights based on confusion created by the ambiguity. This was part of the motivation in the adoption of the expression "open source" in the place of "freeware" (as discussed at section 2.2 below).

2. MIX AND MATCH IN OPEN SOURCE

2.1 The basics

Open source licensing might be touted as the brave free world of software development and distribution but, at its heart, it still depends on the same principles of law as traditional proprietary software licensing.

Ultimately, an open source product carries with it a licence – ie a contract setting out the legally binding rights of the software user and the obligations of the user and the licensor. Similarly, the principles of copyright law and patent law will determine the enforceability of a person's rights and obligations in relation to open

[107] A catch cry of FSF founder Richard Stallman, quoted in Robert W. Gomulkiewicz, "How Copyleft uses Licence Rights to succeed in the Open Source Revolution and the implications for Article 2B" (1999) 36 *Hous. L. Rev*. 179.

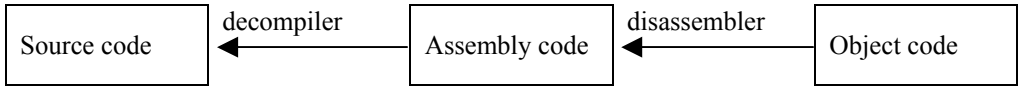
source software – in just the same way as for proprietary software. In the same way as traditional licensing, it will be the traditional courts system (or if the licence instrument requires, which is rare in open source licenses, an arbitration or mediation process) that dictates the outcome of any disputes.

So what is the difference?

In traditional proprietary software licensing, it is only the machine readable or executable version of the code which is made available to the user. The human readable code in programming language (eg C, C++, shell, lisp, assembly, Perl, Fortran, Python, tcl, Java and C#), which is required to enhance, maintain or develop the software, remains undisclosed and the licensee acquires no rights to use the source code.

Parts of the source code may be derived by reverse engineering from the object code, but unless that is authorised by the owner of the software or Division 4A of Part III of the *Copyright Act 1968* (Cth),[108] it will constitute a breach of copyright, and possibly an infringement of patent rights (if there is one), a breach of confidentiality (depending on the circumstances) or breach of licence conditions for the software.

The process for reverse engineering[109] is:



2.2 Open Source Definition

Bruce Perens wrote what the industry regards as the canonical principles defining open source,[110] now adopted by the Open Source Initiative. The principles are as follows:

[108] These are rights of reproduction of licensed software for research and study of the ideas behind the program, for back up purposes, to create independent interoperable products, to correct errors where a commercial correction is not available or for security testing where the test result is not available from another source.

[109] The process of transition from machine readable object code to human readable source code, and the issues in copyright surrounding this, are conveniently set out in Anne Fitzgerald's and Cristina Cifuentes' article *"Pegging out the boundaries of computer software copyright: The Computer Programs Act and the Digital Agenda Bill"* (in Fitzgerald et al *Going Digital 2000*, Prospect Media). It is not proposed to redescribe those principles here.

[110] www.opensource.org/docs/definition_plain.html

Introduction

Open source doesn't just mean access to the source code. The distribution terms of open-source software must comply with the following criteria:

1. Free Redistribution

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. The license shall not require a royalty or other fee for such sale.

2. Source Code

The program must include source code, and must allow distribution in source code as well as 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 preprocessor or translator are not allowed.

3. Derived Works

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.

4. Integrity of The Author's Source Code

The license may restrict source-code from being distributed in modified form only *if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.*

5. No Discrimination Against Persons or Groups

The license must not discriminate against any person or group of persons.

6. No Discrimination Against Fields of Endeavor

The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

7. Distribution of License

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.

8. License Must Not Be Specific to a Product

The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.

9. The License Must Not Restrict Other Software

The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.

Of course, these principles are not legally binding and not all licences of software supplied with the source code implement the entire package of these principles. However, they are a useful benchmark and one used by developers to assess whether a product is "true" open source or not.

2.3 Rights in Open Source licences

Allowing users to have a copy of the source code does not, by itself, make clear what rights the licensee has to use that code – or what rights the licensee has to use, modify or distribute the software generally. These rights are defined in the licence terms. In the context of open source licensing (in the broad sense – not necessarily limited to the scope of the OSI criteria mentioned in section 2.2), the rights that differ from traditional proprietary licensing most commonly fall within the following categories:

- Rights of access to source code (ie the licensee in open source gets the source code);
- Rights of use of the source code. That might be unrestricted or limited to particular purposes (eg for ensuring compatibility of the software with other products; for security checking only; for maintenance; for enhancement and modification; for creation of derivative works or incorporation into another program);
- Rights of copying (eg unrestricted; restricted to the licensed entity; restricted to a particular purpose);
- Rights of distribution eg unrestricted; restricted to related companies or associates; restricted as to form (eg distribute executable code only); restricted as to the conditions to be used when re-supplying or distributing.
- Rights concerning product characteristics (ie warranties) and in relation to rectification of errors (typically open source products attempt to be on "as is"

terms, with no warranty of compliance to a specification or warranty about the absence of viruses, back doors, time bombs etc). This characteristic might not be so different from many proprietary licences, but is universal in open source.

An illustration of how the mix of rights and obligations might be formulated – and the profile of licences from Open Source, through 'Public Source', to the traditional proprietary model is set out in Table 1: "The Open Source Continuum".

2.4 When too much choice is barely enough

Tailoring the mix of rights to suit strategic and commercial objectives (as well as the need to fit with companies' licensing policies and preferred wording for particular clauses) has spawned an enormous number of open source licences. Often licenses have been created to fill gaps or fix problems identified in earlier attempts at a definitive open source licence.

The Open Source Initiative lists 32 different licences that have met their criteria for use of the OSI certification mark. The Free Software Foundation (ie the promoter of GNU) analyses more than 50 open source licences and evaluates them against their views of the "copyleft" ideal (Schedule 1 of this paper has the web references for these).

Taking Red Hat Linux 7.1 (the current version is 7.3) as an example, it has been calculated[111] that there are more than 17 different licence types (as well as public domain software) governing different parts of the source code. The break down on licenses is:

55%	GNU's General Public Licence (GPL)
10%	GNU's Lesser General Public Licence (LGPL)
9.4%	MIT open source licence (MIT)
7.5%	Berkeley Software Distribution licence (BSD)
6.8%	Mozilla Public Licence (MPL)

Because of the heavy use of the GNU terms (either the Public Licence (55%) or Lesser Public Licence (10%)) and because the GNU terms nicely illustrate some of the features and problems of open source licensing, this paper uses that document for discussion.

[111] David A. Wheeler, "More than a Gigabuck: Estimating GNU/Linux's size" at http://www.dwheeler.com/sloc/redhat71-v1/redhat71sloc.html.

Table 1: Open Source Continuum

Copy “left”

All rights reserved

X11, BSD	**Open Source** **(eg GNU General Public Licence)**	**Public Source**
Source & executable	Source & executable	Source & executable
Can modify code	Can modify code	No code modification
Can copy	Can copy	Backup copy
Can distribute	Can distribute	No distribution
Not self-replicating	Self replicating	
No warranty*	No warranty*	Limited warranty*

3. THE SELF-REPLICATION OBLIGATION

3.1 The “copyleft” requirement – derivative works

The reference in Table 1 to “self replication” describes the provision common in open source licences (and in particular in the GNU General Public Licence) which requires distribution of the software and any derivative works on terms compatible with the licence of the original open source program. Some detractors of the GNU model call this a “viral obligation”, whereas others prefer to describe the copyleft requirement as the open source “heritage”.

In the GNU General Public Licence (current version 2, June 1991), the copyleft obligation is expressed as follows:

> You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties *under the terms of this License* [emphasis added] ....

> These requirements apply to the modified work as a whole. 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.

The apparent breadth (and imprecise drafting) of these provisions is frightening to anyone who might not wish to open their own software products to a no-charge open source model of distribution. It is primarily this concern that led to the use of the X11 licence for the class libraries in the Mono Project (discussed earlier in paragraph 4.2). Interestingly, even the creators of the GPL recognised this problem and created the GNU Lesser General Public License in response.

The obligation to apply the same licence conditions to derivative works is binding regardless of the size, importance or value of the original open source code relative to the derivative work. This represents a significant risk to a software developer who uses source code distributed under this model, unless the developer is indifferent to the derivative work having to be licensed on the same conditions – meaning that the entire source code of the derivative work would need to be made available to licensees on the same open source licence conditions.

The second paragraph quoted above (clarifying that identifiably separate software is not covered) introduces the need for well designed and documented processes for code development, so that the origins of those sections of a program which are not derived from the open source software can be readily distinguished from the open source "infected" portions.

This also means that a derivative work may need to have two or more separate licences – one for the open source portion of the software and another for the separately identifiable non-derived portions. Those non-derived portions might be made available on open source terms differing in some respect from the terms governing the "infected" portions – or on proprietary licence terms.

This can be a compliance and marketing headache for software intended for commercial application, requiring an explanation of which licence applies to which portions; click through or click wrap acceptance or execution of two or more licences for one product; and different maintenance and warranty obligations applying to the different portions.

The practical risk of a slip up in compliance with such a regimen is that a user or competitor may be able to insist on disclosure of the source code for the non-

derived portion (ie the independently derived code), where its independence cannot be demonstrated.[112]

A compliance failure might also mean that the licence of the derivative work does not correctly describe the rights of the user (ie it is less generous than required by the open source original licence). That might constitute misleading or deceptive conduct under section 52 of the *Trade Practices Act 1974* (Cth*)* and equivalent provisions in the State and Territory *Fair Trading Acts*.

3.2 Open source impacts for IP policies

Another important aspect of the GNU strategy to overcome restrictions on use of software and its derivatives is to ensure that patents are not used to thwart the intentions of the licence:

> … 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.[113]

This is then reinforced by condition 7 of the GPL which makes the obligations of the GPL paramount over patent rights, namely that such rights must be exercised in a manner consistent with the GNU licence.

4. THE OPEN SOURCE DECISION

4.1 Benefits of the developer community

Despite the shortcomings of the GNU General Public Licence (***GPL***), it has enormous acceptance in the developer community. That acceptance (and the depth of conviction associated with it) must not be underestimated in designing an open source project, where the willing participation of that community (and their *gratis* contribution of coding and improvements) might determine the success or otherwise of the strategy. Not only will such contributions massively reduce development costs and time to market, but they can make the product substantially more secure and more useable. Sometimes those contributions *are* the product.

---

[112] Such an outcome might be sought, for example, by applying the principle of contracts (ie the open source licence of the original portions of code) made for the benefit of third parties (ie the user of the derivative work). In the States of Queensland and Western Australia, this is embodied in statute (section 55 of the *Property Law Act 1974 (Qld)* and section 11 of the *Property Law Act 1969 (WA)* applied in *Westralian Farmers Co-operative v. Southern Meat Packers* [1981] WAR 241), but in other Australian States is derived from common law principles (following the leading case of *Trident v. McNiece (1988)* 165 CLR 107).

[113] Preamble to the GNU General Public Licence.

David Wheeler's work[114] in relation to Red Hat Linux 7.1 illustrates this. He calculates that there are more than 30 million source lines of code (SLOC) in version 7.1, compared to 17 million lines in version 6.2. That represents 8000 person years of programming, worth more than US$1 billion. As already mentioned, more than 65% of the source code is licensed on the basis of the GNU General Public Licence or Lesser General Public Licence.

The support of the open source developer community is strongest when the development task is intellectually interesting or new (or if it would be a blow to Microsoft), but may not be so strong for more mundane programs, where the amount of coding needed is disproportionate the perceived benefit for the developer or where it is thought that the outcome will merely benefit a commercial enterprise (rather than the enhancement of a common good).

For complex software where the cost of development is high, the use of the open source model may discourage development, since the licence terms may make it impractical or commercially difficult to recoup the development cost by licensing the improved (ie derivative) program.

4.2 Balancing commercial interests

An example of commercial pragmatism in open source development is the Mono Project[115] initiated by Ximian Inc in July 2001. Just as Linux sought to clone, in open source, the Unix operating system, the Mono Project is designed to create an open source clone of Microsoft's .NET development platform:

> The .NET development platform is a very rich, powerful, and well-designed platform that would help improve the free software development platform. Just like the GNU project began to clone Unix sixteen years ago, we will be cloning the .NET development platform because it is a great platform to build on.[116]

The runtime components of Mono remain under the Lesser General Public Licence and the programming language (C#) compiler is licensed on the GNU General Public licence terms.

Interestingly, however, the Mono Project has chosen not to use the GNU General Public License for its class libraries, but instead has chosen MIT's X11 license. That decision was made because of the difficulty of securing the involvement of large commercial enterprises in the project if forced to include their work on the same GPL terms.

---

[114] op cit, supra note 5.
[115] www.go-mono.com/rationale.html. This paper does not debate Microsoft's .NET strategy (or its merits or otherwise) compared with products of Sun Microsystems, such as J2EE.
[116] Ibid.

This was a particular issue for producers of software embedded in microchips and devices (like TV set top boxes), where compliance with the GPL has practical difficulties (how can a user be given the ability to change the code embedded in an electronic device, where there is no user interface for this?) as well as being commercially unpalatable.

The principal difference stemming from use of the X11 licence is that commercial contributors who use the class libraries to create improvements or derivative works will not have to make their final source code available (as they would under the GPL).

The decision was rationalised by Miguel de Icaza, Ximian's co-founder and chief technologist for the Mono Project, in the following terms:

> So this doesn't prohibit Intel from making an optimized Intel-only version (of Mono) that they wouldn't release to the world – that is a downside. It does worry me a bit, but the advantage is getting large contributors to the project.[117]

Such pragmatism, and its impacts for continued support of the development community, must be carefully thought through to make sure it does not cost the support of the development community.[118]

Licensors contemplating open source also will need to analyse the implications on direct licensing revenue (and whether other revenue models are available, for example in providing associated services such as technical support and implementation services) and the impacts for their strategic and competitive position. The trade off might be between the commercial strengths of different intellectual property assets – the source code, on the one hand, and strength of brand for service provision, on the other.

Open source might lead to a product becoming widely accepted and a *de facto* standard, and the benefits of that in a particular case might be enormous for services or other products of the licensor.

4.3 Whose source is it?

For a software licensor, the analysis of whether to use an open source licence should include some analysis of the components of the software and their

---

[117] Quoted in Wired at http://www.wired.com/news/technology/0,1282,50037,00.html.

[118] An interesting example of this is where AT&T in the early 1990s sought to better profit from its UNIX program, increasing licence fees and litigating in respect of the residual UNIX components used in the Berkeley Software Distribution (***BSD***) version of UNIX. The litigation was settled, but the BSD development community replaced the AT&T code – isolated AT&T – to solve the problem. For the interesting history, see Andrew Leonard's "BSD Unix: Power to the people from the code" www.salon.com/tech/fsp/2000/05/16/chapter_2_part_one/print.html

respective origins. One might think it should go without saying, but it is necessary to be certain, before disclosing the source code, that:

- The licensor's own staff developed the source code within the scope of their employment;
- Any code developed by consultants to the licensor was done on terms that either expressly passed ownership (ie all intellectual property rights) to the licensor or which granted the licensor rights to distribute the source code in the manner contemplated;
- Any code which is licensed to the licensor was acquired under license terms that permit distribution of the source code in the manner contemplated (and all conditions of those licenses are complied with, eg requirements about notices and labelling); and
- No commitments have been made to a third party (eg existing customers) that source code would not be disclosed or that it would be kept confidential.

4.4 The user's choice

For a user, it is not only the initial cost of acquisition that is relevant, but also the cost of ownership – impacted by maintenance costs, system administration time and downtime risks and costs. Depending on the software, the open source model might increase or decrease costs of ownership.

Even where many community developers contribute to the enhancement of open source software, users will need to carefully examine the origins of improvements made and the quality of the improvements to determine their suitability and effect for the user's systems. That involves maintaining programming skills, time and cost.

If it transpires that a person contributing source code has infringed the intellectual property rights of someone else (ie the source code was pirated), a commercial enterprise using the end product has legal exposure to the true owner of intellectual property rights. That the open source licence conditions expressly disclaim any warranty (including a warranty that the software does not infringe anyone's intellectual property rights) is only part of the problem, since pursuing the developer for the infringement in most cases would be financially (if not legally) futile.

Many who argue the benefits of community software development also argue that the model will result in more robust or more secure software. That is not necessarily so, since it assumes that those accessing the code will use the code only for improvement – and not to find security holes or to write tools for use in hacking systems which are using the software. In business, it cannot be assumed that people are universally well motivated and not evil or mischievous.

Some argue that code and system crackers have not found it difficult to reverse engineer source code for closed source products (or to write viruses and hacking

tools using published application programming interfaces) and that a security strategy based on non-disclosure of source code is flawed.

Much of the debate around security is centred around bug fixes, rather than the security implications of underlying software architecture; specifically the role of open source in relation to analysis of the security of the architecture and user choices about security and access.

The US National Security Agency (**NSA**) has done extensive work in relation to Linux, arguing that open source for operating systems plays a critical role in relation to security.[119] The resulting security enhanced Linux (or SE Linux) is designed to allow the operating system to serve the security choices of the user, rather than having to rely on the security decisions of a particular software vendor, allowing the user to dictate mandatory access controls. However, SE Linux does not embody a suite of other security features such as security audit or system assurance, which nevertheless are important elements for the user's security strategy. The NSA acknowledges that a complete security solution still requires considerable work to add that functionality before SE Linux will be a "Trusted Operating System" suitable for meeting a particular government or corporate user's requirements.[120]

Whatever the security virtues of open source, it requires just as much vigilance of system administrators in tracking known bugs and implementing available fixes to minimise the risks of unauthorised system access or attacks. Recent analysis by Mi2g suggests that Linux based web server systems are increasingly being targeted by system crackers, with a 27% increase in successful system attacks in the first 6 months of 2002, with the success partly because of system administration issues.[121] Whatever the reality about security advantages of open source software, it may be completely inappropriate for a highly security sensitive user to use open source. In some markets the perception of such a security risk (regardless of reality) may be sufficient reason to exclude use of open source software for critical systems. However, there are many examples of open source software which is so widely used that it becomes mainstream and thoroughly acceptable for commercial use.

---

[119] See the 2 January 2001 press release at www.nsa.gov/releases/selinux_01022001.html and information about SE Linux at www.nsa.gov/selinux/index.html.
[120] See response #19 at www.nsa.gov/selinux/faq.html.
[121] Mi2g report of 11 July 2002 at http://mi2g.com/. Press report at http://www.zdnet.com.au/newstech/os/story/0,2000024997,20266696,00.htm

5. NO WARRANTY PROVISIONS

5.1 "As is" conditions

The GNU General Public Licence (current version 2 June 1991) and most other open source licenses offer the software on an "as is" basis, with a clause stating that there is no warranty of the software.

In the GNU General Public Licence, that is expressed as follows:

> Because the program is licensed free of charge, there is no warranty for the program, to the extent permitted by applicable law. Except when otherwise stated in writing the copyright holders and/or other parties provide the program "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the program is with you. Should the program prove defective, you assume the cost of all necessary servicing, repair or correction.

As discussed below, there may be some real problems which arise under Australian law in the scope of protection such a clause offers a licensor. That is not peculiar to open source licences, but is such a fundamental feature of open source development that it is worthy of mention.

5.2 Excluding implied warranties and conditions

As discussed in this section, an exclusion of implied conditions and warranties may not be effective in Australia, even when the governing law of the open source licence is a State of the USA or some other place. The application of the implied conditions involves some complexity where the supply is for no monetary charge, but in some cases the implied conditions and warranties still apply.

In Australia, there are conditions implied into contracts of supply by the *Trade Practices Act 1974 (Cth)* (***TPA***) and State and Territory *Sale of Goods* legislation. The absence of a written contract or express terms will not necessarily prevent a finding that a contract does exist, into which the conditions and warranties may be implied.

The conditions implied by the TPA (sections 66 to 74) are:

- In relation to goods – warranty as to title; warranty of quiet enjoyment; warranty that the goods are free from encumbrances; a condition that supplies by description will comply with the description; a condition that the goods will be of merchantable quality; a condition that the goods will be fit for any purpose the consumer expressly or impliedly makes known to the supplier; and a condition that goods supplied by reference to a sample will correspond to the sample.

- In relation to services – a condition that the services will be rendered with due care and skill and that materials supplied in connection with the services will be reasonably fit for purpose; a condition that the services will be reasonably fit for any purpose the consumer expressly or impliedly makes known to the supplier (unless it is unreasonable for the consumer to rely on the supplier's skill or judgement).

There is some debate in Australia about whether software (by itself) constitutes goods or services (and therefore which of these sets of implied warranties are relevant). A sale of a complete computer system (hardware and software) is regarded as a sale of goods,[122] but there is no settled authority about a mere licence of software. If software is distributed over the internet, without the sale of a physical medium of supply, the supply is merely a grant of rights and does not have any of the usual characteristics of a supply of goods. In such a case, it might be argued that a pure supply of software is a supply of services, rather than a supply of goods – avoiding, in particular, the implied warranty of merchantability.

Of course, it is another issue to determine what a warranty of merchantability or fitness for purpose might mean in practice for a supply of software (say a session tracking tool) that is intended only for further development and testing and integration into a much larger whole (eg the Mono Project), which will then also be tested and improved before being ready for commercial use.

In other words, a single developer participating in an open source community development of a product like Mono or Linux, has little practical exposure or risk from an implied warranty of merchantability or fitness for purpose.

However, once the product is in a form ready for commercial implementation and use, a provider who licences the compilation of software contributed on this basis by hundreds of developers, would have good reason to consider the implications of an implied warranty of merchantability and fitness for purpose – and to consider how liability in relation to those warranties might be excluded or limited in a way that is legally effective.

The implied TPA conditions cannot be excluded by a provision in a contract[123] and any attempt to do so is void (although liability can be limited to resupply or the cost of resupply in some circumstances, discussed shortly). A provision stipulating a governing law of a place outside Australia, will not prevent the conditions implied by the TPA applying.[124]

---

[122] *ASX Operations Pty Ltd v Pont Data Australia Pty Ltd (No 1)* (1990) 27 FCR 460. *Toby Constructions Products Pty Ltd v Computa Bar (Sales) Pty Ltd* [1983] 2 NSWLR 48. *St Albans City & District Council v International Computers Ltd* [1996] 4 All ER 481 (CA).
[123] Refer to section 68 *Trade Practices Act 1974* (Cth).
[124] Section 67 *Trade Practices Act 1974* (Cth).

The TPA conditions apply to supplies to consumers. This is defined in the Act[125] so that a supply will be to a consumer only if:

- The price of the goods or services (the amount paid or payable) does not exceed AUS$40,000; or
- Where the price is greater than AUS$40,000 – if the goods or services are "of a kind ordinarily acquired for personal, domestic or household use or consumption".

A great deal of open source software is supplied for no charge. The TPA states that where services are acquired (defined to include "accepted") other than by way of purchase, then the price for the purposes of the $40,000 threshold can be determined by the available price for purchase or, if not available for purchase, the value of the services. If the value is zero (a give away), then the supply is for less than the $40,000 threshold.

It is not necessary that the software be sold or purchased, as long as it is supplied or accepted.[126] However, the TPA conditions and warranties will be implied only where the supply is "in trade or commerce" – words which the courts have given a wide and generous meaning.[127] Where software is supplied by way of gift, not sale, this requirement nevertheless would be satisfied if the software supply is part of a commercial dealing or if the supply is connected (even indirectly) with advancing or protecting the commercial interests of the supplier.[128] That may not be too difficult to satisfy, particularly where a licence is associated with a commercial supply of ancillary services (like software support or documentation).

Taking the GPL Public Licence (***GPL***) provision as an example, it does state that the exclusion of warranties in the first sentence of the clause (set out above in section 5.1) is made to the extent permitted by applicable law, but that override (even assuming it is effective to prevent the first sentence being void) does not necessarily apply to the remainder of the clause. It is possible that the entire exclusion provision in the GNU General Public Licence will be rendered void since there is no provision allowing invalid parts of that clause to be severed from the rest. Red Hat's licence for Red Hat Linux version 7.3 recognises this issue.[129]

The *Trade Practices Act* allows suppliers to limit their obligations under the implied conditions to resupply or the cost of resupply,[130] but that is subject to an overriding test of whether it is fair and reasonable to allow liability to be limited in this way. In any case, the GNU General Public Licence does not purport to limit liability in this way.

---

[125] Section 4B.
[126] *Clarke v New Concept Import Services* (1981) ATPR 40-264, at 43,348 per Davies J.
[127] For example, Deane J in *Re Ku-ring-gai Co-operative Building Society (No. 12)* (1978) 36 FLR 134 at 167.
[128] *Fasold v Roberts* (1997) 70 FLR 489.
[129] www.redhat.com/licenses
[130] Section 68A.

Also, because its terms require that any distribution of the software (or derivative works) be on the same terms, distributors of GNU licensed software face difficulties in correcting these problems – except by expanding the licence terms with additional exclusions and limitations of liability. There is an argument that such attempts are incompatible with the original GNU conditions and unenforceable, but that point does not seem to have been taken up by anyone.

6. EXCLUDING LIABILITY OF AN OPEN SOURCE SOFTWARE SUPPLIER

6.1 The GNU General Public Licence

The GNU General Public Licence (current version 2, June 1991) attempts to limit liability of the software supplier in the following way:

> In no event unless required by applicable law or agreed to in writing will any copyright holder, or any other party who may modify and/or redistribute the program as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties or a failure of the program to operate with any other programs), even if such holder or other party has been advised of the possibility of such damages.

For the reasons discussed below, it would be far better for the licensor to have an exclusion clause more broadly and unambiguously drafted, and which takes account of the laws applying in Australia and the limitations to liability permitted under Australian law.

Of course, with the self replicating nature of the GNU licensing model, the opportunity to redraft the licence to rectify these problems is severely constrained – except by expanding the licence terms with additional exclusions and limitations of liability. This leaves those in the supply chain potentially exposed with liability under implied warranties (discussed above in section 5.2), liability in negligence (see section 6.2 below) and liability for loss of profits and loss of revenue (see section 6.3 below).

Also, a disclaimer clause by itself will not erase liability of a supplier of open source software for misleading and deceptive conduct under the Australian *Trade Practices Act 1974 (Cth)* (discussed in section 6.4 below), but there is not much that the licence conditions could do about that.

6.2 Exclusion clauses and Australian courts

Australian courts will give an exclusion clause its natural and ordinary meaning but, if there is any ambiguity, it will be construed against the person seeking to rely

on the clause.[131] The courts look at the provision as a whole and, if the exclusion attempts to limit liability for the very purpose of the contract, it will need to be clearly and unambiguously drafted to survive challenge.

Where there is ambiguity, courts will read the provision narrowly (ie less protection for the supplier of the software), including if a wider reading would be irrational or unjust[132] or conflict with another provision in the licence.[133]

If liability for negligence is not expressly excluded, the courts may read down the exclusion clause so that liability for negligence is not excluded.[134] It is best therefore to specifically exclude liability for negligence. There is a real risk that the GNU General Public License does not exclude liability for negligence of the licensor.

Anyone using software licensed under the GPL to create derivative software also would be obliged to use the GPL for the whole derivative work, and therefore may be exposed to damages for negligence in relation to their work.

6.3 Consequential losses

The law in Australia about clauses which attempt to exclude liability for consequential loss is complex, derived from various court decisions in Australian and the UK. A detailed review of that case law is beyond the scope of this paper, but in summary:

- Losses which flow directly from a breach of warranty or obligation are not "consequential losses" or "indirect losses";[135]
- Courts have been willing to generously define direct losses, to include things like loss of revenue, loss of profits,[136] and the cost of labour and materials needed to remedy a breach;
- An exclusion of "consequential losses (including loss of profits or data)" can be read narrowly by a court so that liability for loss of profits or data is excluded only

---

[131] The leading case is *Darlington Futures v Delco Australia* (1986) 161 CLR 500, endorsed in *Nissho Iwai Australia v Malaysian International Shipping* (1989) 167 CLR 219 and applied in *Kamil Export v NPL* (1996) 1 VR 538.

[132] *Carlingford Australia v EZ Industries* (1988) VR 349.

[133] *Shoard v Palmer* (1989) 98 FLR 402.

[134] *Bright v Sampson and Duncan* (1985) 1 NSWLR 246 held that and exclusion of "all liability" or "liability for any loss" did not exclude liability for negligence, but additional words of "whatever its cause" would have been sufficient. In *GL Nederland (Asia) v Expertise Events* (BC9901003, 16 March 1999), an exclusion of "liability whatsover for damages" was sufficient to exclude liability for negligence, even though negligence was not specifically mentioned.

[135] *Croudace Construction v Cawoods Concrete Products* [1979] 2 Lloyds Rep 55 (Court of Appeal).

[136] *Deepak Fertilisers and Petrochemical v Davey McKee* (London) [1988] CILL 1448 (Court of Appeal).

where it is a consequential loss.[137] In other words, if the loss of profits or data results directly and naturally from the breach, then an exclusion in that form might not prevent a claim succeeding.

Clauses which seek to exclude liability for losses of revenue, profits, data and business opportunities must therefore be very carefully and particularly drafted.

The GNU General Public licence (and a great many traditional proprietary software licences for that matter) does not adequately exclude liability for loss of revenue or profits. Also, there is a risk that the exclusion for loss of data will be read to apply only when that is a consequential loss and not when the loss of data is a direct and natural result of the breach (for example, a breach of an implied condition or warranty).

6.4 Misleading or deceptive conduct

Section 52 of the *Trade Practices Act 1974* (Cth) prohibits a corporation in trade or commerce from engaging in conduct that is misleading or deceptive or likely to mislead or deceive. Similar prohibitions existing in State laws.

The prohibition has been applied in an enormous range of circumstances including pre-contractual negotiations,[138] advertising or promotional material, labelling and "small print" cases.[139] It can apply when there is silence in circumstances where relevant facts should have been revealed.[140]

If an exclusion clause lessens the misleading nature of what has gone before (eg by correcting some previous misleading conduct or statement), it might have some effect to lessen liability.[141] In most instances, however, this will not be the case and the contractual exclusion of liability will be disregarded in determining liability.[142]

7. OTHER THINGS THE GNU GENERAL PUBLIC LICENCE DOES NOT DO

7.1 Services related to software supply

There are some things the GNU General Public Licence (and many other open source licences) does not seek to do. For example, there is no provision in the GPL

[137] *Pegler v Wang* [2000] BCL 218.
[138] eg *Bevanere v Libidineuse* 59 ALR 334. Discussed in Terry, "Consumer Protection for Business Interests: The application of section 52 of the Tr*ade Practices Act* to commercial negotiations" (1987) 10 UNSWLJ 260.
[139] eg *TPC v Optus Communications* (1996) ATPR 42-478 and *Britt Alcroft (Thomas) LLC v Miller* [2000] FCA 699.
[140] eg *Henjo Investments v Collins Marrickville* (1988) 79 ALR 83.
[141] *Benlist v Olivetti Australia* (1990) ATPR 41-043.
[142] *Clarke Equipment v Covcat* (1987) 71 ALR 367.

for the supply of services ancillary to supply of the software (such as pre-supply assessment of client needs; installation and integration services; training; post supply support; and on-going maintenance services).

For a commercial supplier of software (or a service provider for whom the software supply is ancillary), these things may be the main source of revenue for a transaction and would need to be supplied under a separate services or support agreement.

7.2 Missing contract provisions

There are many contractual provisions which a licensor or user may wish to include in a software licence used in Australia, but which the GNU General Public Licence (and others like it) do not include. For example, there are no clauses:

- Applying principles of proportionate liability where a user is partly responsible for a loss suffered (in response to the decision in Astley v Austrust);143
- Setting out the governing law of the licence and dealing with the parties' submission to the relevant court's jurisdiction;
- Applying non-litigation dispute resolution processes (such as mediation or arbitration);
- Allowing provisions which are unenforceable to be severed or read down without affecting other provisions in the licence;
- Regulating assignment of the licence;
- Dealing with confidentiality of information exchanged in the context of the transaction;
- Dealing with costs and taxes (including GST);
- Dealing with insolvency or termination;
- Stipulating the licence as the entire agreement of the parties; or
- Other contractual "boiler plate" clauses to aid interpretation and enforcement.

The absence of some of these things is a function of the expectation that the licence will not be used in the context of a commercial transaction or involve the payment of any fee or other consideration. However, that will not always be the case and the licence (without the addition of such provisions) would not be appropriate in a business context.

---

[143] (1999) 197 CLR 1 (4 March 1999). Statutory reforms have been made in response to this decision to ensure that a claim for a breach of contractual duty that also is a breach of a tortious duty will nevertheless be subject to apportionment for contributory loss: see definitions of "wrong" or "fault" in section 14 *Law Reform (Miscellaneous Provisions) Act 1955* (ACT), section 15 *Law Reform (Miscellaneous Provisions) Act 1956* (NT), section 8 *Law Reform (Miscellaneous Provisions) Act 1965* (NSW) and *Statutory Duties (Contributory Negligence) Act 1945* (NSW), section 5 *Law Reform Act 1995* (Qld), section 25 *Wrongs Act 1958* (Vic), section 2 *Wrongs Act 1954* (Tas) and section 3 *Law Reform (contributory Negligence and apportionment of liability) Act 2001*.

Other things in this list are just as relevant in non-commercial simple licensing and their omission could prove detrimental to reliance on the document and its enforcement.

8. OTHER ISSUES IN OPEN SOURCE LICENSING

There are a number of issues in open source licensing which are unresolved, but are not peculiar to Australia. This paper does not attempt to deal with them, but they are mentioned here for reference:

1. Role of the open source project manager

There is usually not any particular clarity about the legal position of an organisation which co-ordinates the programming efforts and management of results of a community effort in open source development.

2. Enforcement of copyright and licenses

When a compilation of software with hundreds of authors exists, there are some difficult issues about whether there is a compilation copyright held by the project manager (ie independent of the underlying copyrights in the individual programs) and who would have standing to enforce the copyright in the compilation. Also, there are difficult procedural issues about how a large number of copyright owners would be joined in an infringement action or be served notice of the proceedings.[144]

There are a great many difficulties about what remedies would be appropriate also, given that (i) there are multiple contributions to the total work of differing size, functionality and relative importance; and (ii) contributors have not required payment or royalties for their work.

3. Making licences binding

Licences of software in open source projects are rarely signed by a licensee – the software is transmitted via the internet with notations about the licence terms under which it is contributed. The way in which that is done varies widely and it is not necessarily the case that the software can be accessed only after the user has accepted the terms of the relevant license in a formal "click through" process.[145]

That raises doubts about whether the licence terms will be binding at all. It does not necessarily help that the relevant licence terms contemplate a click through method of acceptance (as does the GNU General Public Licence), if there is no implementation of that method of acceptance before the software can be downloaded.[146]

---

[144] Discussed in Shawn W. Potter "Opening up to Open Source", 2000, 6 Rich. *J.L. & Tech*. 24. Some of these issues are mentioned in Paul B. Lambert "Copyleft, Copyright and Software IPRS: Is Contract still King?", E.I.P.R 2001, 23(4), 165-171.

[145] Like the "I agree" mechanism accepted by the court in the *Hotmail Corporation v Money Pie* case C98-20064 (N.D Cal, 20 April 1998).

[146] Refer to the factors the court took account of in *Rudder v Microsoft* [1999] O.J. No 2778 (Ontario SC, 8 October 1999).

9. CONCLUSIONS

The expression "buyer beware" requires a broader application in the open source model: not only does the product itself need to be considered against that principle, but a user also needs to consider:

- The implications for the on-going costs of ownership of the software (in view of the availability of support services and the need to vet improvements offered by community developers);
- The implications for security of the systems on which the software will be used (particularly whether community access to the source code might allow security flaws to be found and wrongfully exploited, just as much as that knowledge might instead be used for common good) – as well as how that issue might be perceived (regardless of reality) by stakeholders and investors; and
- Implications for the user's strategies for acquiring and benefiting from intellectual property – in particular, whether the demands of the open source licence might have a deleterious impact for patent activity of the user or the commercial benefits available from works which use or are derived from the open source software.

Developers also need to consider the commercial, strategic and competitive implications of using the open source model. Sometimes, the model may offer enormous benefits in reducing development costs or extending or accelerating market penetration. In other cases, the reversal of traditional intellectual property rights might not be commercially attractive, denying the opportunity for licence revenue.

Where open source is used, the terms of the licence are fundamentally important. Conditions should be tailored to suit the particular needs of the licensor and crafted to achieve identified commercial objectives. Use of a particular existing form of licence can carry with it risks that liability is not effectively excluded and that other necessary contractual provisions are omitted.

Schedule 1

Selected References

A word of caution with references: Many of the writings on open source software are from the perspective of developers who regard the social ideal of shared software and the developers' community above all other ideals (such as securing a revenue model), without acknowledging some of the legal difficulties with open source licensing (such as those canvassed in this paper) or the concerns raised by others about security issues. They are valuable for what they are, but should be read with that in mind.

Open source licences
Lists of licences with links to the licence conditions:

GNU http://www.gnu.org/licenses/license-list.html	The Licences are categorised according to whether they are compatible with GNU's General Public Licence (for combination of GPL compatible software from different sources for on-supply), whether a "free software" licence or whether "copyleft" licence (but not GPL compatible).
Open Source Initiative (*OSI*) www.opensource.org/licences/index.html	A useful site, including details of use of the OpenSource certification mark and approval processes to become entitled to describe software as OSI Open Source Certified Software. Else where on the site is the canonical definition of Open Source and the nine principles of it.

Publications

Dennis M. Kennedy, "A Primer on Open Source Licensing Legal Issues: Copyright, Copyleft and Copyfuture", 20 *St. Louis U. Pub. L. Rev*. 345. Contains a concise history of open source licensing and a US perspective on some of the major licenses and principles.

Shawn W. Potter "Opening up to Open Source", 2000, 6 Rich. *J.L. & Tech*. 24. Contains some interesting discussion of legal (US) and industry issues.

Raymond, Eric S, *The Cathedral and the Bazaar*, available at http://tuxedo.org/~esr/writings/cathedral-bazaar/. This was an early (and evolving) articulation of the different philosophies of open source (and its colourful, eclectic, interactive and cooperative character – the bazaar) and proprietary, autocratic and highly managed approach of traditional commercial licensing (the cathedral). A manifesto of the true believers.

Newsforge, The Online Newspaper of Record for Linux and Open Source, www.newsforge.com/

For a useful collection of links see http://www.dwheeler.com/oss_fs_refs.html

Security

John Pescatore, "Microsoft Sends Mixed Signals About Software Security", 13 May 2002, Gartner, at http://www3.gartner.com/DisplayDocument?doc_cd=106790.

Kenneth Brown, "Opening the Open Source Debate: A White Paper", June 2002, Alexis de Tocqueville Institution, withdrawn from publication but made available at http://www.adti.net/html_files/defense/opensource_whitepaper.pdf See also the discussion of this paper at http://www.wired.com/news/business/0,1367,53124,00.html

NSA's discussion about SE Linux at www.nsa.gov/selinux/index.html

Schedule 2

Short Glossary

A useful reference for other expressions is http://whatis.techtarget.com/.

Be careful with expressions like those described below, since industry usage of the expressions varies greatly. Defining exactly what you mean in a licence, website or software documentation is the best approach, to avoid any ambiguity about usage of a word.

Using one of these words does not necessarily convey a precise meaning and could lead to misuse of the software or dilution of legal rights based on confusion created by the ambiguity. With that warning in mind, here is a limited glossary for those starting out in open source.

BSD or Berkeley Software Distribution	used narrowly, means the open source licensing model developed in the late 1970s and early 1980s out of the University of California at Berkeley by Bill Joy (later a co-founder of Sun Microsystems) in their efforts at improving Unix. Unlike the GPL model of the Free Software Foundation, there is no express obligation to return improvements and derivative works back to the common pool on the same terms as the original open source licence, although that is part of the underlying philosophy of the licence. Used widely, BSD refers to a number of licences of a similar kind, including FreeBSD and Open BSD.
Copyleft	The principle that a modification or derivative of a work (software, manuals etc) should be distributed on the same basis as the original work, ie with the same freedom to copy, modify (therefore with access to the source code) and distribute. Some call this a "viral" obligation, because it causes the self replication of the copyleft principal for all distributed and derivative works. Not all Freeware is copyleft – a licence might allow the user to distribute modified or derivative work on terms that do not grant these freedoms.
Free	See Freeware
Free Documentation	See Freeware.
Freeware or **Free Software**	Does NOT necessarily mean the software is supplied at no charge. What it refers to is the freedom of use – usually, freedom to distribute and modify the software (therefore with access to the source code) and to use pieces of the software to make new programs. However, the expression does not necessarily mean that there are no restrictions on use or distribution. Freeware could be on terms that require any on-licensing or any distribution of derivative works to be on the same terms as

	the original software (ie a "copyleft" requirement). Similarly, "free documentation" used in open source parlance generally refers to a particular level of freedom to copy, modify and redistribute it, with or without the payment of a fee.
FSF or **Free Software Foundation Inc**	The organisation formed for the GNU Project and now an advocate of open source free software, found at www.gnu.org/. FSF promotes 3 forms of licence: • The General Public Licence; • The Lesser General Public Licence; and • The GNU Free Documentation Licence.
GFDL or **GNU Free Documentation Licence**	The GNU licence of the Free Software Foundation for the licensing of manuals and other documentation associated with Free Software – allowing freedom to copy, modify and distribute, whether or not on payment of a fee. Found at www.gnu.org/licences/fdl.html.
GNU	A recursive acronym for "GNU's not Unix", pronounced "guh-new". Originated in 1984 as a project of the Free Software Foundation (see www.gnu.org/) to develop a Unix-like operation system as freeware, although it is now an organisation with wider advocacy for open source licensing.
GPL or General Public Licence	The standard GNU Project open source software licence of the Free Software Foundation Inc, found at www.gnu.org/licences/gpl.html.
LGPL or **Lesser General Public Licence**	The software library version of the GNU Free Software Foundation Inc's General Public Licence – permitting linking of the software library into non-free programs, but without the requirement of the General Public Licence that the entire derivative work be distributed on the same free software basis as the original library. It is found at www.gnu.org/licences/lesser.html.
Open Source	is where software is distributed in both source code (ie human readable programming language) and in the executable (machine readable) form. Open source does not necessarily imply that the code is made available on a "copyleft" basis or that it is "freeware". For the canonical definition of Open Source and the nine principles of it, originally written by Bruce Perens, see www.opensource.org/docs/definition_plain.html
Public domain software	usually refers to software where its creator allows it to be copied, used, modified and distributed with no attached restrictions or limitations as to ownership or payment of fees. Sometimes this is by an express waiver of rights, but more often because of what can be implied from the method of it being made available, the absence of any assertion of legal rights and other surrounding circumstances.

Shareware	often is used in relation to trial software distribution. The software is shared, sometimes with a built in disabling time clock (ie a validity period or drop dead date), to allow the user to examine it before deciding whether to buy a full license. If some of the functionality is disabled it is called "liteware". Shareware is not necessarily open source, copyleft or freeware.
Viral obligation	A licensing obligation that distribution of the software – and of software which is derived from the original software – be on the same basis as the original code's licence, ie the means by which "copyleft" principal is effected in a licence. The expression is not one the FSF and other open source proponents find flattering.

Chapter 5

Security with Free and Open Source Software

PROFESSOR WILLIAM J (BILL) CAELLI
Head, School of Software Engineering and Data Communications,
Queensland University of Technology

INTRODUCTION

This paper considers the debate on security in relation to the *"open"* versus *"closed"* software environment. It discusses concepts of trust in this context and a particular case of one open source, high-security operating system under development. It then evaluates emerging international standards that may have an impact upon the choice of such trusted systems over what could be argued as being less secure, commodity options.

In considering the basics of security in relation to complex computer software systems, there is a need to discuss concepts of trust and then the application of this to so-called "trusted" or "trustworthy" systems, the term "trustworthy" implying a lower level of overall confidence and security functionality. A particular example of software research and development aimed at the production of such trusted software systems and one under consideration at QUT is known as *"SELinux"*. This is a 2 year-old security enhanced version of Linux that has been developed by and is available, under open source licencing conditions, from the National Security Agency (NSA) in the USA.[147] From a management perspective, however, the situation that is really likely to emerge could be termed the *"why choose trusted systems"* problem. The emerging imperatives for this need to choose come about in response to two new international standards. These are:

- International standard IS-17799, known as the "Information Security Management (ISM)" standard, and

[147] NSA claims: "End systems must be able to enforce the separation of information based on confidentiality and integrity requirements to provide system security…. Unfortunately, existing mainstream operating systems lack the critical security feature required for enforcing separation: mandatory access control. As a consequence, application security mechanisms are vulnerable to tampering and bypass, and malicious or flawed applications can easily cause failures in system security."
See: http://www.nsa.gov/selinux/ [11 November, 2002]

- International standard IS-15408:1-3, commonly known as the "Common Criteria" document, which is concerned with security and evaluation techniques for creating and assessing so-called "Trusted Systems".

How these two standards impact on the whole problem of open versus closed security is examined in this paper. Finally, we will consider what influence, if any, software licensing procedures may play in this area.

THE MOTOR VEHICLE ANALOGY

There's no doubt about it; cars are gradually becoming computers on wheels.[148] However, put simply, we want our cars to be more reliable than our desktop computer systems. When driving on a high speed highway, the very idea of "rebooting the car engine" while operational is simply unacceptable. In the car case, we all want safety and security inside with confidence that we are travelling in a trustworthy environment, as indicated in recent Mercedes Benz car advertisements. At the same time, we have to acknowledge that on the outside there are other drivers who can metaphorically "attack and damage" the particular vehicle that we are using.

The recent IEEE article by Berger, about the forthcoming generation of motor vehicles, sets out a series of propositions and consequences related to the incorporation of computer systems into motor vehicles. The propositions are that:

- Failure can ruin a purchase, with the customer left unhappy;
- Failure can cause injury and death, even more true as microprocessors and allied embedded systems occur in practically everything;
- Servicing is costly;
- Users give the product almost zero maintenance; and,
- Complexity is causing more failure points.

So reliability and testability cannot be an afterthought and should come very early on in the design process. Formal development tools are needed. Communication, particularly amongst the developers themselves, is critical. Such communication is needed up and down the whole development chain if one is going to have security in any product or system.

Of course the important concern in the context of this paper and the information technology industry, in particular the software sector, is just how will this apply in an "open source" environment. In other words, what is the relationship "chain" in open source software systems development that allows for enhanced security and thus trust to be solidly incorporated? Is it any different, better or worse than the in the so-called "closed" or "proprietary" software systems case?

[148] Berger, I. *"Can you trust your car?"*, IEEE Spectrum, April 2002.

IT'S NOT THE NET, IT'S THE NODES

Today, the major problem we are facing in relation to the security of information systems and data networks is not completely based upon the underlying and connecting "Internet" itself. However, it could be argued that users like to think that the problem is the data communications infrastructure of the Internet and indeed the press usually reports information security breaches as an "Internet problem". In actual fact, it must be submitted that *"it's not the 'Net, it's the nodes"*. In other words, major security threats may be assigned to the computer systems that form the "nodes" of the global Internet; the hosts/servers, the end-user client systems and computer based switch/store-and-forward computers. Thus, in assessing the background to illicit activity the following questions may be posed.

- How many cases have been successfully prosecuted in relation to detection, capture and prosecution of those involved in so-called "line tapping" activities in relation to the Internet?
- How many "rogue" Internet Service Provider (ISP) personnel have been arrested for illicitly intercepting data line activity or store-and-forward message banks?

The answer in both cases appears to be "very few" at the most. In general, case history points to problems with vulnerability of the nodes to even unsophisticated attacks. The number of prosecuted and/or reported cases involving actual "line tapping" of data in transit appears to be extremely small compared to those related to computer systems penetration. The majority of that penetration, either by illicit access or by insertion of Trojan horse / virus programs, has occurred because of difficulties and limitations in the operating system and/or middleware, such as data base servers, web browsers, etc., used on the systems forming the nodes; the servers, clients and switching systems.

INFLUENCES ON THE NEED FOR SECURITY

Why is security suddenly so important? We now have an Information Security Management Standard (IS 17799) being adopted by both Federal and State Governments in Australia and elsewhere in the world. This standard may be considered as a definition of a minimum set of responsibilities on management to ensure that information systems public and private enterprises provide to any end user are fit for their purpose. At the same time International Standard IS 15408, the *"Common Criteria"* standard, sets out corresponding security requirements for the developers and producers of computer and allied information technology products and systems. It defines concepts of security "functionality" required by users of systems and, then, the assurance or "evaluation" levels needed to ensure that users may trust the security functions provided. However, few commodity products have been evaluated and tested under this standard and, where this process has been undertaken, they have attained only the very basic level of security functionality and evaluation.

Understanding of these standards requires that managers and information technology professionals alike have some information security background in the form of education, training and experience. Unfortunately this may not often be the case and it may be unreasonable for the chief information officer (CIO) in an enterprise to be able to assess the security status of products and systems employed in the enterprise. For the home and small business user, such an evaluation is simply impossible as it is for any other complex system, e.g. a modern motor car braking system, etc.

In many cases this means that a manager usually calls upon external contractors to perform security assessments on their system giving rise to the sudden growth in companies specialising in this arena, with, again, reasonable questions being raised about their competence through education, training and experience in the security area. Such an assessment will normally cover IT products, systems and operations since the ISM covers all of these three matters.

In summary, ISM is important since it is rapidly becoming a "due diligence" standard for information system managers in both public and private enterprises. However, whether or not enterprise management has the ability to judge installed IT systems against that standard is a contentious question.

THE "PERIMETER SECURITY" VIEW

An alternative view to information systems security at the computer system level has emerged over the last twenty years, the period of the development of the commodity personal computer system. The situation may be summarised in the following statement *"Hey hold on – don't worry about the computer system itself – we'll protect it by putting file walls and anything else we need around that system and that will safeguard it when connected"*. We call this attitude one of total dependence upon *"Perimeter Security"*. However, we must now acknowledge the failure of such a perimeter security paradigm as the only basis for overall protection of enterprise information systems. It was, incidentally, not the paradigm of the mainframe or minicomputer era of the around the 1960s to the 1980s. The concept of embedding security into the computer system itself was clearly agreed, as evidenced by the influence of the "MULTICS" system, developed at the Massachusetts Institute of Technology (MIT).

Perimeter security problems and limitations are increased as moves are made to high bandwidth or "broadband" data networks. Imagine if we had to filter every single packet moving through a particular multimedia conversation or the like, established through a "web services port". Imagine if every packet had to go through a complex application level rule implemented in some sort of next generation intelligent, filtering, fire-wall. Performance degradation is enormous and obvious. In actual fact what we must likely would do is to forget all about such security and "punch a hole" through the whole fire-wall system and establish unfiltered, direct connection.

So broadband is now starting to spell the end of what we know as total dependence upon "perimeter security". Indeed, *perimeter security should never have become a primary defence system*. It is unfortunate that it became that. Indeed, it became such a primary defence mechanism at the systems level in the late 1980s onwards as manufacturers abandoned well-designed and implemented information security parameters at the operating system and even at the hardware levels in the 1980s to meet cost and market demands, as they now readily admit. With the possibility of over 1 billion PCs having being sold and global interconnection to the Internet a reality, the problem of "backstitching" security into the PC's operating system could be seen as almost an unsolvable problem.

PUBLIC KEY INFRASTRUCTURE (PKI)

In addition, we have also seen the failure in widespread and consistent adoption of PKI (Public Key Infrastructure)[149] based digital signatures for verification of the source of software systems and data downloaded from servers on the Internet. Adoption of the need for assurance of the actual source of software packages in an open Internet environment has now been readily accepted. If nothing else I know that if my software system fails I can say "well at least I know where it came from – or I hope I know where it came from".

The problem is, however, that I really have no assurance at all that what I installed is what I thought I installed. A good example of this can be attested to by those who downloaded and used the "Kazaa" system to do "peer-to-peer" information sharing. They can suddenly find that they have a few little "Easter eggs" inside their system, i.e. program features and activities that they simply didn't know about, want or even permit. Verification of the source of a program offers no security information related to its content, quality or behaviour. In this sense, then, digitally signed software packages only offer forensic evidence to end users in cases of security violation caused by downloaded packages.

NATIONAL INFORMATION INFRASTRUCTURE PROTECTION (NIIP)

We are also starting to see political pressure emerging, particularly in larger Western countries, and under the auspices of the Prime Minister, the Minister for Information Technology, Communications and the Arts and the Attorney General in Australia, towards the need for protection of the national information infrastructure. In this regard, national information infrastructure protection (NIIP) will have to become a shared responsibility between the private and public sector

[149] The Public Key Authentication Framework is a framework for the generation, distribution and management of public key certificates. These certificates bind the identity of users to their public key material in a trusted and legally based manner. See: National Office for the Information Economy, "Gatekeeper Frequently Asked Questions", [http://www.govonline.gov.au/projects/publickey/faqs.htm#12] 11 November 2002.

as even Government and defence enterprises make use of privately owned information technology infrastructure or outsource information service provision to the private sector. This particular pressure may start to put new legal obligations on directors in targeted vertical industries in Australia and overseas. For the first time, we may actually start to see explicit legislative instruments move through the United States Congress in relationship to the added responsibilities of directors or owners of what we would call critical infrastructures in the United States. This could also extend to other countries.

In this regard, the future appears to be concerned with the concept of "web services". Such "web services" will again become a "backbone" for critical information systems in society. This is the background against which we have to consider where we are headed in relationship to information security and, of course, this is again relevant to the open source movement.

SOME BASIC INFORMATION SECURITY PROPOSITIONS

Security Architecture and Implementation

First, secure systems involve more than commitment to bug fixes and software quality. On web discussion groups about information or system security, concentration has been on bug fixes, particularly so-called "buffer overflow" problems. Computer security is far more than that. Indeed, robust computer security, as demonstrated by the MULTICS research activity, clearly requires that secure systems must be ones that *tolerate* such software quality problems without degradation of overall system security.

Thus, in relation to the open source security debate, the central theme of this paper, the discussion is much more related to the understanding and development of the following matters:

- existence of an underlying system security architecture, with appropriate mechanisms
- robust system design,
- security engineering,
- hardware interaction,
- continuous, trusted enforcement of security functions, and
- assessment and evaluation of the underlining security architecture.

In other words, it is not just bug fixes and software quality that is the matter in hand. The question is: *"Does the software system implement and support a comprehensive security architecture, in cooperation with the underlying hardware, which is well defined, providing the appropriate features that are consistently enforced?"*

SYSTEM LEVEL DOCUMENTATION

Almost 40 years ago, in the era of mainframe systems, machines such as the IBM System 360 and 1401, Control Data 6600, and others were all supplied with complete "system programmers documentation" or "system programmers manuals". These tomes used to occupy whole bookcases. They contained the detailed specifications needed for a program to interact with the operating system and by which computer professionals could maintain, repair and optimise the performance of the operating system itself. Fullness and completeness were the targets of this particular documentation. The idea that "undocumented" system features (calls, etc) could exist and be tolerated was simply non-existent. The concept that security could be found in not documenting existing system features was simply not considered a reasonable approach.

APPLICATION "VIEWS"

There is an important difference between application program development today and that of some twenty years or more ago. The difference may be summarised as that between an application's "middleware view" and its possible "system view". A good example of this can be gleaned from both Microsoft "Windows" and LINUX based computer systems. An application developer creates a computer program by making "calls" to some form of "middleware" sub-system, such as a graphic screen service emulating a "desktop environment" to display information on a screen or to accept information from an input device, or to a "web browser" sub-system. In the Microsoft case, this may be the so-called "Win32" application programming interface (API) while in the LINUX case it may be the so-called "X-Windows" interface or higher level sub-systems.

However, it is usually possible for the application program to make direct "calls" to services offered by the operating system itself, so-called "system calls", through an "operating system API". In many cases today, these "system calls" are simply not documented and the secrecy of their existence and proper usage may even be carefully guarded by the software supplier. In simple terms, open source simply prevents this from happening in principle since all source code for the operating system, its sub-systems, such as the graphics screen or "desktop" system, and the like may be perused and all interfaces determined and fully documented. From a security viewpoint, the need for full documentation of all system calls must be regarded as being of extremely high priority since these calls may be responsible for notable security actions at the operating system level, e.g. change the "privileges" of an application program.

SOFTWARE COMPONENTS

We are faced with a future based upon the usage of *"software component"* libraries. All software suppliers are moving in that direction. The idea is that in the near future programmers and other IT professionals will be accessing whole application systems and libraries of systems, based upon "reusable component libraries", and integrating them into overall information systems to meet enterprise needs. Moreover, it may be that non-expert users may be capable of performing such functions themselves with little to no training, the ultimate "end-user computing" paradigm. Trust will now have to be placed not only in the computer's hardware and basic operating system but also in the myriad of library functions used to construct final application programs. To use the car analogy again, IT users will become "car drivers" selecting a basic car and formulating desired options, with no knowledge at all of underlying principles, engineering structure, robustness or the like. Dependence upon security assurances by the supplier will be total.

However, this does mean that new levels of security and control will be needed at the operating system level. The phrase *"B means Business"*[150] refers to the requirement for so-called "mandatory" security controls in operating systems. This is in line with what is known as the set of requirements for *"B level"* systems in the USA's 1980s *"Trusted Computer System Evaluation Criteria (TCSEC)"* or *"Orange Book"* terminology. In other words, as we move to component systems we are going to have to "up the anti" and an operating system will need to enforce the security policy of an enterprise in a far more reliable way simply because the applications that form that enterprise's information system will be constructed from components whose real source, security functionality and evaluation status may not be known.

"PALLADIUM"

The *"Palladium"* may be remembered as a big and old show palace in London as well as a precious metal. However, the term is now used by Microsoft to describe its next generation hardware and software based security subsystem for PCs that incorporates a digital rights management (DRM) system for such data "content" as Hollywood films and music.[151] This creates a special "secure zone" on the motherboard of the PC. It is also a de-facto admission that the PC now connected to the Internet is not secure for Web commerce. Now, interestingly, Microsoft has

---

[150] Caelli, W. *"B means Business",* Proceedings of the National Information Systems Security Conference, Baltimore, USA. 1995.

[151] *"Palladium is Microsoft Corporation's code name for an evolutionary set of features for its Windows operating system. Combined with a new breed of hardware and applications, these features will give individuals and groups of users greater data security, personal privacy, and system integrity."* As per John Manferdelli, General Manager Windows Palladium Business Unit [http://www.microsoft.com/presspass/features/2002/jul02/07-01palladium.asp]

announced that it will reveal the source code for the operating system level support software for this Palladium innovation.

"We will be publishing the source code because people will need to trust this", said Mario Juarez, group product manager for the Palladium project at Microsoft. *"To get people to believe in what is happening in that little piece of code is critical".*[152] But revealing the source code does not assist with complete specification details of the underlying hardware that forms the Palladium structure, particularly if that hardware structure is eventually incorporated into the main CPU chip of the PC, i.e. the Intel Pentium's replacement. In a strange way, a Microsoft representative has already made the case for open source in the above statement!

HISTORY AND TECHNOLOGY

"E-Government"

It is always valuable to examine the history of technology. This may help to shed light on one question of note in this paper, i.e. *"Why is it that information system security is becoming quite important"?* One answer lies in moves towards so-called *"electronic"* or *"e-government"*. It is common around the world for some governments, for example, to have *"whole of government"* agreements with suppliers, such as Microsoft just as with IBM in the past. Indeed, government information activities may be wholly *"outsourced"* to third parties. Almost by default, commodity, consumer PCs will be used as the vital user interface to these critical information systems. Such companies will then automatically be involved in moves towards e-government systems, for example. So security of their systems is critical.

Consequently I pose a question asked in 2001 relating to the Microsoft Windows-2000 operating system:

> Can someone please tell me why a fault in my soundcard driver has to crash my system? I mean, really. Think about it.[153]

This is a real problem since essentially what in the past may have been regarded as an esoteric operating system level problem becomes a real world concern for desktop users. Security, in the form of "availability", can be radically affected by the simple addition of a sound element to a desktop PC used in a critical information system in any public or private enterprise.

---

[152] Robert Lemos, "Microsoft to reveal Palladium source code", News.COM, 24 June 2002, [http://news.com.com/2100-1001-938973.html]

[153] P Viscarola, "Peter Pontificates: I Dream of … A new Version of Windows", *The NT Insider*, OSR Open Systems Resources Inc, Vol 8, Issue 4, July-August 2001

Secure Applications

An important problem emerging throughout the world in the IT industry is the wrong idea that all information security problems can be solved by putting security in the application. This is wrong – it can't be done. A recent paper from IBM researchers stated the proposition as follows:

> Hardware on which applications run must be secure as must be the operating system and run time environment in between, while offering a reasonable API for application developers … Applications cannot be more secure than the kernel functions they call, and the operating system cannot be more secure than the hardware that executes its commands.[154]

Fundamental logic indicates that applications cannot be more secure than the functions that they call and it must be of concern that an important IBM research paper has to once again emphasise a basic fact of computer science. The problem is, however, that IT consumers may be convinced that this could be so and that work by the vendors of computer systems can be supplanted by application developers. This is particularly true as we move to software component libraries, as mentioned above. Now, in turn, the operating system cannot be more secure than the hardware that executes its commands.

That is particularly important in relation to the Intel central processor unit (CPU) chips that are used in IBM compatible PCs and elsewhere. For example, it is not widely understood that the memory addressing capability of the Intel Pentium chip is not 32 bits but is actually 36 bits. There are other computer hardware architecture parameters that need to be closely examined in this family of Intel CPU chips since these basic functions have a fundamental bearing on the overall security assessment of computer system. The point is that most users are totally unfamiliar with these parameters while expert attackers may be. This hardware architecture concern can best be illustrated by an examination of the Microsoft XP system.

The following statement is taken from a Microsoft published manual that relates to the development of support software for the Windows 2000 and the XP operating systems:

> Although each Win32 process has its own private memory space, kernel-mode operating system and device driver code share a single virtual address space …Windows 2000 doesn't provide any protection to private read/write system memory being used by components running in kernel mode. In other words, once in kernel

[154] Dyer et al, "Building the IBM 4758 Secure Coprocessor", *IEEE Computer*, October 2001 – is there a more complete reference eg page number??

> mode, operating system and device driver code has complete access to system memory space and **can bypass Windows 2000 security to access objects.**[155]

This presents a major security risk to any manager of information systems who plans to use Microsoft Windows 2000/XP based end user workstations or desktop systems. For example, suppose that I'm setting up a new e-government system for internal and external usage by employees and contractors. Digital signatures and virtual private networks (VPN) will all be used. Now, imagine that we have got a specific group that has decided to innovate with "teleworking" or "telecommuting". We have an employee working at home on his or her PC or even laptop system. A virtual private network (VPN) structure is used to get back to the office information system. Imagine that at some time one of his or her children start using the machine for some games and that they are logged into what they consider to be a popular games site on the web. What happens? Very, very simple. The game site says:

> "Hey – you need the new exciting 5.1 surround sound simulation system"
> *"OK",* says the child
> *"Please down load it now"*
> The child does. The PC's operating system then asks the child
> *"Do you wish to install this – Yes or No?"*
> Of course, the child should say "no" because this is an important system to install. But he or she naturally clicks the "Yes" button.
> What has happened? The download has inserted a device driver into the operating system.

As soon as that driver is installed *security is totally compromised*. The driver can now "address" everything, bypassing all operating system and application security functions. It can now talk to every part of that particular machine including bypassing all the encryption on the virtual private network system and any of the cryptographic service providers that perform all the digital signature functions. In particular, compromise of the cryptographic subsystem done in this way means that the evidentiary status of such techniques as "digital signatures" and allied "digital certificate" usage is now severely lowered and may be rendered useless. Of course, the end-user should not have loaded the driver but, more importantly, the design of the operating system should never have allowed a device driver to be capable of such security bypass.

"Charlie Chaplin"

How did we get into such a state of poor operating system basic architecture and implementation? Very, very simply: because in the early 1980s we were never meant to be where we now are with usage of the then "personal computer" or PC.

---

[155] D. Solomon and M. Russinovich, "Inside Microsoft Windows 2000", Third Edition, Microsoft Press, Redmond, Washington, USA, 2000

The idea here was that the PC was meant for the end user. It was called a *"personal"* computer, not an "office system", workstation, enterprise system or the like. It was not linked up to anything. There was no network. It was stuck on your desk. Today's equivalent example would be the unconnected "Personal Digital Assistant (PDA)". The idea was that the system was to be used by the individual person who took complete responsibility for it and suffered alone any consequences of its failure or any breach of security.

Through the PC, IBM advertisements explained, the user liberates his or her own information activities from the "dreaded, white-robed scientists" in the corporate or government computer centre, pre-occupied with their "IBM System 370s", or whatever it happened to be in those days and taking an interminable time to create any new application program that was needed. Thus IBM used the "Charlie Chaplin" figure to promote the IBM PC; the representation of "everyman", the "battler" overcoming the pressures and oppression of the "establishment" while maintaining that spirit of independence and fortitude necessary to survive.

This *"Chaplinesque"* symbolism is an important factor in understanding the complete disinterest in any form of information security technology in relation to the PC by its manufacturers during the first 15 or more years of its widespread existence and acceptance by business; the place where it was not intended to be!

Indeed, this concept of "liberation" from "information control" at the centre of a corporate information system was again emphasized in the famous 1984 "Superbowl" advertisement for the then Apple MacIntosh computer with its spirit of defiance exemplified by the smashing of a "big brother" figure. All of this meant that security and the PC became themes that were *totally opposed* to one another and this problem has existed for over twenty years.

Thus we can pose the question: *"Where are we today?"*

FIVE ALTERNATIVES FOR PC SECURITY

In relation to the PC of today, whether it can be used as a host/server, end-user client system or switch/controller, we have a problem in evaluating the comparative security worthiness of "open" versus "closed" systems. We can identify five particular areas of interest in a complex series of problems. In this regard, we really need the analytical and classificatory skills of the legal profession to try to specify and categorise the complex matters involved as we start to discuss security in this environment.

Alternative 1 – Proprietary / Closed Systems

The first and most obvious practice is to totally control access to the source code of the PC's operating system, as exemplified by the Microsoft Corporation stance, on the understanding that such an action is necessary, or even highly beneficial, to

preserve the security of the system. This does not mean that some users do not have access to the source code but rather that it is not freely available to any end-user or consumer of the product as a matter of course or under reasonable terms and conditions. In the car analogy case, there is no readily available and complete "workshop manual". In addition, even those who do have access to the code would not normally be involved, as was the case in the early days of the mainframe systems of the 1960s and 1970s, in employing full time "system programmers" to create and deploy bug fixes, performance and/or feature enhancements or the like to the operating system itself.

In some ways this could be called "security by secrecy or obfuscation" and it includes the case of:

- no ready access by any purchaser to source code for usage and modification or even for perusal, nor
- availability of full and proper system level documentation.

This path seems to have been advocated and confirmed by Mr J Allchin of Microsoft in testimony to the USA's Department of Justice antitrust case in Washington, DC. Essentially, he is quoted as stating that *"too much disclosure of technical information in the wrong areas would benefit hackers and create more opportunity for virus attacks"*. In the Microsoft case the disclosure of Microsoft proprietary information in this area was supposed to be of national security significance. Of course, such an argument indicates that any security architecture and its implementation may be subverted through access to information on its structure. This violates one of the most fundamental rules of security for commercially available systems, as illustrated by the MULTICS work in the 1970s and the commercial cryptography paradigm, i.e. security does not depend upon secrecy of mechanisms but rather on a limited data element set.

The rule simply states, as for the case of encryption systems, that knowledge of the structure of the system must not violate its security. In encryption, this means that knowledge of the cipher algorithm by an opponent may be assumed but not knowledge of the cryptographic keys required. In the operating system and computer hardware case, a similar rule would indicate that while the architecture is known, individual controls, such as segmentation- bounds registers, cannot be over-ridden by application or device driver software or the like.

From an end user's point of view this "closed systems" approach requires full and complete trust in the operating system supplier and makes contractual obligations highly one-sided. The vendor of the software system has knowledge, expertise, information and data that are simply not available to the customer. This is like a car manufacturer sealing the bonnet/hood of a car to prevent the customer/user from perusing the engine and effecting any modifications or repairs. In another sense, an operating system vendor in this arrangement could easily have a major competitive advantage over application developers should it wish to also enter that application

market since it may possess information totally unknown to the application developer that may benefit the supplier in many ways.

In arguing this way, a vendor is largely presenting a "time and expertise" argument in relation to the capabilities of any potential attacker of the system. However, reverse engineering technologies such as test harnesses, in circuit emulators of central processor units, both hardware and software based, disassemblers, de-compilers, etc. have all become widely known and available in the IT profession and global expertise in this area is on the rise.

A good example of the problem of trying to depend upon proprietary or closed systems is exemplified by the famous Microsoft "Xbox". Those who have been following the Xbox saga know that it is a closed system, or it was, until this month (July 2002). A paper from MIT gives a good overview and perception on how to reverse engineer the Xbox.[156] Now, why would anyone want to a reverse engineer an Xbox? One of the main reasons the researcher at MIT claims for doing it, was to investigate privacy issues related to those who used the Xbox for online tasks. This is highly relevant since Microsoft recently talked about the Xbox for online gambling and online gaming applications. It is known that information, such as the serial number of an Xbox console, installed electronically, is probably accessible to the kernel of the Xbox's operating system. Now, what happens to this information when an Xbox is plugged into the Internet? Well, who knows? Encryption is used to secure various parts of the Xbox but the nature of information relayed to, for example, Microsoft's projected online game service, is totally unknown. So, the possibility of utilising a closed system for massive monitoring of people could be claimed to be quite enormous, as documented fully in the MIT paper.

In summary, then, an argument against closed systems for security is that given time and resources, both becoming available, a closed system can be penetrated. Time and expertise can defeat *"security by obscurity"*. For example, there is a very popular reverse assembler program named *"IdaPro"* from Europe. This quite good reverse assembler can handle a number of popular central processor units, not just those from Intel, and it is available now.

Thus, available technologies are getting better, from test harnesses to reverse assemblers to decompilers. Indeed, as an aside, the use of a so-called "test harness" to shed light upon the structure of a software system through structured testing of its actions, has been broadly seen as legally quite permissible. It does not involve disassembly or decompilation and leaves the target software system, in its binary form, intact. However, the end result may be the same.

Finally, use of closed systems by enterprises has a number of interesting aspects to it. Under the standard for Information Security Management, IS 17799,

[156] Details are available the following web site at 23 January 2003:
http://www.xenatera.com/bunnie/proj/anatak/xboxmod.html

management is responsible for timely and effective creation of required security processes to meet overall information system security needs. We now have to recognise that if we adopt closed, proprietary systems there will be an inability of the end user to repair a system if needed after a successful attack caused by a system "bug" and total dependence upon the supplier has to be clearly evaluated and documented. Moreover, the end-user or manager may have absolutely no ability to judge the security status of the underlying operating system or hardware and is compelled, in IS 17799 terms, to assign any statement in these areas to the vendors of the systems.

Alternative 2 – Full System Documentation Supplied

This alternative mirrors the mainframe situation of the 1960s and 1970s when complete "system manuals" were made available to "systems programmers" for maintenance and enhancement activities. Could it be made legally binding upon manufacturers that system documentation be made available to the end user and that it be made full and complete? Could we say that there is a requirement that no undocumented features exist? In this sense, the consumer can be made knowledgeable about just what he or she has purchased. A car analogy helps again. So called "workshop manuals" have existed for a long time and contain the details needed for the maintenance of the car and even for its enhancement with add-on features. Is this a valid security alternative?

Questions that have also to be determined include:

- Is that system documentation reasonably priced and readily available, and
- Is there unlimited usage on such documentation, apart from obvious copyright matters.

From the end-user point of view, in determining security parameters, an end user organisation would have to possess the expertise to make use of such documentation and to assess the security status of the organisation's information infrastructure with contributions from that documentation.

Alternative 3 – Source Under Agreement

In an access to source code agreement between the supplier and the purchaser/licensee, the user has access to the source code of the pertinent software systems. In this case, it can be argued that responsibilities and liabilities as to information security matters then lie with the user in that the user could have determined any underlying weaknesses in security architecture and implementation. In turn, such weaknesses could then have been assessed in relation to the overall information system security requirements.

This is an interesting prospect particularly here in Australia as we look at the standard IS-17799. The Australian and New Zealand version of it give

responsibilities to managers. But is it sufficient to be able to say that since we've signed up to have access to source code under agreement and therefore have availability, we will be able to fulfil our obligations? The end-user enterprise would have to assign expert personnel to assess such source code. That might be satisfactory for large organizations but for the home or small business end-user it is unrealistic. The inexpert home user particularly, say in the e-government environment, will have nothing to do with the source code of, say, Microsoft's *"Windows XP"* or the application system sitting on top of it. The user simply "sees" an "end user environment" or relevant application operating on the computer system. Thus, at an individual level there is a problem. In summary, the value to the consumer is dubious in the case of source code availability under agreement.

However, there is an interesting aspect in this for the emerging "software component libraries" of the future. We can pose the question:

- "How will I know what I'm getting if I access a component library?", and
- "Whose responsibility is the guarantee for security and safety of that library?"

Alternative 4 - Open Source

In this case, we are talking about the equivalent of the workshop manual in the car industry. For example, I can walk out and buy a Holden Commodore. I can go to a shop and buy a freely available workshop manual. I feel free to open up the bonnet or hood over my engine, undo the engine, pull it apart and use my workshop manual to obtain the information I need in order to "hot it up" or to cool it down or to change it. I can do this in anyway I feel like. I have that right. Thus this case refers to the condition of ready availability of the source code to a software system at any time, by anybody, at a reasonable cost and without any form of onerous agreement being required.

I submit that this analogy sheds light on precisely how the "open source" issues we know today came into existence in relation to the PC / workstation / server environment. It could be submitted that IBM, Microsoft, and other companies have become resistant to such openness because of hardware and firmware disclosure decisions made at the start of the 1980s, not because of software. Indeed, without that disclosure it could be argued that the PC revolution as we know it could not have occurred and that companies such as Microsoft would have remained as small, specialised vendors to IBM. Essentially, the disclosures made possible the "IBM PC clone" industry and the rest "is history".

The interesting reason for all this was that the IBM PC Model 1 in 1981 was made available with an optional, low cost book called a "Technical Reference Manual". This manual from IBM contained the complete schematics for the whole *"motherboard"* of the PC, its fundamental architecture and hardware/firmware construction. It contained the source code for what was called the "BIOS" or

"Basic Input/Output System" that allowed external devices to be connected to the machine and for any required operating system to readily "converse" with the underlying hardware in a generalised and convenient way. It contained full hardware diagrams and hardware schematics. An interesting side effect of this was that the structure of the IBM PC was able to be evaluated by IT professionals.

Of course in the case of the car if I bought my workshop manual and pulled apart the V8 engine on my General Motors car, I really don't automatically assume that I have a right to build another General Motors V8 engine that is an exact carbon copy and to market it in competition. This was even true in the early 1980s as "clone PCs" emerged. But I could resort to other parts of the law, that offer protection of my adapted or derived General Motors V8 engine. By having knowledge of my V8 engine, for example, I could remove two cylinders and call it the new "Super 6" engine. The "ideas" of the original engine have been used but the manifestation is different. This was the case with the PC clones and the birth of such companies as Compaq and others. So we are back to intellectual property concerns and interpretation of pertinent law.

Alternative 5 - Freeware

This is generally agreed to be the real concept of "open source". A user has the right to obtain, adapt, modify, use, pass on, etc. the source code to the software system under liberal licensing agreements. The source code may even be published in an accessible way with no charges applied for such access. But can a user make money on the code used? If commercial activity is desired with such freeware, what different arrangements need to be made, if any?

THE OPEN / CLOSED DEBATE

Which of these systems helps us in relationship to assessment of the security of an overall information system itself which incorporates products or systems subject to agreements as outlined above?

The debate as to just what the "rights and wrongs" are in relation to security assessment and the open versus closed source code question moves between the two obvious limits. These are simply that making the source code of software generally available can create a security problem of varying intensity versus doing this is essential for the creation of trust. Depending on just who is making the argument the "intensity" at each end is seen as highly variable. However, there is an opinion that, from a security and technical point of view, closed versus open source argument is largely irrelevant. It has to be emphasized that in this technical view it is assumed that the statement as to security assurance becomes the responsibility of the information system management and not the system manufacturer or vendor.

Recently Anderson (LEMO-02) has argued that, if security is measured by the failure rate of software in relation to security "holes", then an analysis similar to the "mean-time-before-failure" paradigm used in other engineering areas indicates that open and closed software systems may be expected to be about the same. He is quoted as saying that *"other things being equal, we expect that open and closed systems will exhibit similar growth in reliability and in security assurance"*. The problem here is that the concentration appears, in the reports, to be centred around software *"bug reports"* rather than the discovery and analysis of more fundamental design and construction flaws related to system security which may alleviate such problems. In other words, does any higher level software system fully support or fully use any underlining hardware or system software features? In this regard the question must then be posed: "What is the ability of the application oriented IT professional or the information system's managers or user to assess the underlying operating system itself?

SECURITY IN THE COMPUTER HARDWARE

Segmentation Hardware

In the not so distant past the main information security facilities and their enforcement were based upon the computer's hardware architecture and implementation. The Multics proposal, mentioned above, of the 1960s to the early 1980s proposed a full, hardware based security architecture. The Multics system, while itself not widely accepted into mainstream mainframe computer products offered to the marketplace, did lead, however, onto the design of the Intel 286 processor and to the Intel "Pentium" processor that we have today. For example, the buffer overflow problem which has been one of the main problems in computer security for at least the decade of the 1990s, was largely solved by the Multics computer science people a long time ago by use of a concept of "memory segmentation" architecture. Not only did the segmentation architecture enforce proper memory management at the hardware level but it also allowed for the separation of data and program code at the hardware level such that data could never be "executed". However, what was happening at a deeper level with Multics was the realisation that computer architecture for reliable information security had to assume that computer programs normally did have "bugs" that could affect the total operation of the system and thus this major factor had to be allowed for and overcome. This also added further hardware based security mechanisms to the Multics structure, such as the famous *"protection ring"* technology, not considered in this paper, but also implemented in the Intel iAPX-286 CPU and beyond to today. This enabled the separation of program types into varying levels of security trustworthiness.

Now, the segmentation hardware architectures proposed in the Multics system, as a mechanism for added computer security, were essentially carried through into the Intel processor environment from the early 1980s and the Intel iAPX-286, used in the IBM PC-AT computer. However, unfortunately that architecture is effectively

"turned off" when examining the operations of current commodity operating systems, including Linux and Microsoft's Windows systems. The problem of buffer overflow threats to system security, while being solved a long, long time ago, still exists today since system software developers just simply did not want to know about this advanced segmentation architecture that, as they wrongly thought, imposed additional design thought and implementation complexity on software development.

So, here is an example of how freeware or open source will let us assess just one level of security architecture existing within the system. In the closed system environment we cannot assess whether or not the full security features of the hardware are properly and fully utilised in the operating system structures. We are totally dependent upon any statements of the manufacturer in this regard, as evidenced by the Windows XP device driver case mentioned above.

Judgement on the eventual use of the full security features of the Intel Pentium or beyond has to be reserved even though they are fully available to be turned on by commodity oriented operating system developers. However, how do you know to what extent these features are turned on or turned off, used or ignored, correctly or incorrectly supported and so on? An answer depends upon the willingness of a vendor to provide complete information on this or by changing the source code of the operating system itself yourself, the open source/freeware case.

THE "RAINBOW SERIES" OF THE UNITED STATES' GOVERNMENT

Thus we have had an architecture for computer security for a long time. The problem was one of assessing such structures so as to give some assurance as to the trustworthiness of a computer system. Now 20 years ago the United States then National Bureau of Standards (NBS) published the first of what was to be known as the *"Rainbow"* series of books. The first of these books, each one usually given a nickname after the colour of its cover and thus leading to their nomination together as the *"Rainbow Series"*, was the 1983 *"Trusted Computer System Evaluation Criteria (TCSEC)"*. It is worth noticing that this concept of defining a series of publication that outline the concepts and methodology for determining the security of computer systems has now been standardised internationally as international standard IS-15408, also known as the *"Common Criteria"*.

The important thing about all this is that besides saying that we have to have security functionality in a computer system that can be used to create trusted applications and so on, we have to have an enterprise system security policy that is reliably and continuously enforced by the system. Such an assurance must come from the vendor unless, of course, we can assess the operating system and any relevant "middleware" ourselves, a process of "evaluation" which requires the pertinent source code and expertise.

Assurance

But we have to make sure that we have such security assurance. It is required to make any real statement in relation to the IS-17799 security management standard for "real-world" information systems. Assurance means that hardware / software mechanisms can be independently evaluated to provide sufficient evidence, and then confidence, that a system enforces security requirements. Moreover, such evaluation also provides evidence that the system is reliably and continuously protected against tampering and/or unauthorised changes. Indeed, the orange book gave us the parameter set by which we can judge the security architecture and resulting security enforcement in any system. Now, the "Orange Book" sets out six simple but basic and pervasive requirements for any secure computer system. These may be summarised as follows:

POLICY:
1. Security policy must exist and be defined to the system
2. Marking of all entities must be possible

ACCOUNTABILITY:
3. Identification of all entities
4. Accountability for all actions

ASSURANCE:
5. Assurance that security features exist and are reliable
6. Continuous protection is provided.

Can an end user perform such an assessment in a closed environment? The answer is obviously "no" and he or she must depend upon a trusted third party's evaluation or on statements and guarantees provided by the vendor.

End users normally have no reasonable way of making that assessment. Users of information technology products and systems, particularly where these are imported from another country, may become totally dependent on security assurances provided in a completely lopsided contractual arrangement with a software vendor. The end user may have no power at all to make even the vaguest statement as to the security status of an operating system or the underlying hardware incorporated into an enterprise's information system. With growing system complexity it is highly likely that such a situation will become much more obvious over the next few years. The important point to note from the "Orange Book" experience is that highly secure operating systems and related computer hardware could be created and sold to meet the "Orange Book" standards in the 1980s at the highest levels. The problem is that such high trust operating systems, including the *"GEMSOS"* operating system for common, low cost hardware such as the Intel 286 based systems, never entered the commodity computer marketplace and they were largely ignored outside the military and government markets.

CASE STUDIES

Sesame

This paper finishes with two relevant studies which, it may be submitted, further make the case for the open source environment in relationship to information and computer/network security. "Sesame" was a European program that acknowledged the security problems with commodity operating systems. The project concentrated on operating systems because it was recognised in Europe they really are the important factor in overall information security. Unless information systems are created around a secure operating system environment, everything else can be forgotten. As stated earlier, you cannot create secure applications on top of an insecure operating system. It is complete nonsense to try to say that you can.

The SESAME project aimed at the enhancement of basic operating system functions to incorporate so-called "role-based access control" services and related mechanisms. It also anticipated the use of "Smart Cards" and like tokens for the incorporation of user "profiles" that could be reliably enforced by the operating system itself.

Secure Linux

More interesting has been the involvement of the USA's "National Security Agency (NSA)" in relationship to the security of operating systems. This involvement, publicised some two years ago, is a major pointer to the problem of security in commodity operating systems.

Basically, a paper from the NSA by Losocco[157] et al, clearly stated that the computer industry has not accepted the critical role of operating systems and security. This statement was as follows:

> "The computer industry has not accepted the critical role of the operating system to security, as evidenced by the inadequacies of the basic protection mechanisms provided by current mainstream operating systems. The necessity of operating system security to overall system security is undeniable; the underlying operating system is responsible for protecting application-space mechanisms against tampering, bypassing, and spoofing attacks. If it fails to meet this responsibility, system-wide vulnerabilities will result.
>
> The need for secure operating systems is especially crucial in today's computing environment."

---

[157] "*The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments".*
by Peter A. Loscocco, Stephen D. Smalley,Patrick A. Muckelbauer, Ruth C. Taylor,S. Jeff Turner, John F. Farrell, National Security Agency, USA. Available at *http://www.nsa.gov/selinux/doc/inevitability/inevitability.html.*

The authority of researchers at the National Security Agency of the USA must be taken into consideration in relation to any risk statement pertinent to the security of computer and data network systems. The above statement is unequivocal. The operating system is responsible for protecting application mechanisms against tampering and bypass, a major problem in the case of device drivers as indicated for the Microsoft Windows 2000/XP case mentioned above. Industry has not accepted this responsibility.

So what did the NSA researchers do? Linux was chosen as a platform for the work because of its open development environment. What does that basically mean? It means that we can now demonstrate strong security functionality that has been made successful in a mainstream, commodity level operating system. Once again the choice of Linux for this work is clearly explained by the NSA at its web site, as follows:[158]

> Linux was chosen as the platform for this work because its growing success and **open development environment** provided an opportunity to demonstrate that this functionality can be successful in a mainstream operating system and, at the same time, contribute to the security of a widely used system. Additionally, the integration of these security research results into Linux may encourage additional operating system security research that may lead to additional improvement in system security.

Now this work becomes critical as we start to talk about national information infrastructure protection (NIIP), particularly as governments worldwide move to e-government services. In this regard, Standard 17799 places obligations on public sector managers to make binding statements about the security of their underlining systems.

So, what is the reason for the SELinux development? The NSA project team made the following summary statements in relation to their activities:

- The operating system is the right place for security,
- "Mandatory access control" is the correct model for security in the Web/Internet world,
- Access to "objects" must be controlled by an enterprise policy administrator,
- Users / programmers / processes cannot change security policy, and
- All users / uses are mediated by a trusted service in regard to the policy

The benefits of such an approach may also be summarised as:

- safe execution of untrusted software, such as that obtained from Internet based sources,
- limited scope for potential damage due to successful penetration of the system,
- separate "environments" for users and developers,

[158] Statement available at the following web site: http://www.nsa.gov/selinux.

- an insulated/controlled development environment,
- isolated testing facilities,
- clear separation of policy and enforcement activities, and
- the advantage of an "open source" community able to improve and widely distribute and influence security architecture.

SUMMARY AND CONCLUSIONS

Computer systems that reliably and securely implement the *"mandatory access control"* concept, whereby the policy is set up by management not by the programmer as with most commodity operating systems, are required for safe and secure connection to the global Internet. This is a minimum standard, particularly for "e-government". In this regard the SELinux activity sets out some minimal parameters for judging the status of security in relation to current Internet connected systems. SELinux is a "statement" of the required level of security for commercial / government usage, from a trusted and acknowledged authority. The NSA has given a clear statement of the non-performance by industry in the security area, and provides a demonstration of trust requirements to the IT industry.

In essence, government is demonstrating by example with a system that:

- has widespread availability, with appropriate education / training,
- is highly relevant in the government server market,
- is not limited to one LINUX distribution, and
- complies with the "GNU Licence" arrangement with associated cost advantages.

The GNU licence has the advantages that end-user organizations have the legal ability to modify and upgrade security as needed to meet risk assessment under IS 17799 for example. This may be favoured over unknown or vague vendor guarantees in the area. From a market perspective, safety and security never have been market driven, e.g. seat belts in cars, fire extinguishers in the office or home, smoke detectors in homes, pool fences around swimming pools for child protection, etc.

In this regard the GNU licence offers many advantages. An enterprise has the legal ability to deploy, install and on-sell secure operating systems and related sub-systems such as compilers, middleware, browsers, Internet protocol "stacks", etc) as needed. These may be used within that or associated enterprises or sold or given to other enterprises or individuals. In this way a person acting as a manager or director of that enterprise may be able to meet any information security obligation or equivalent duty as a "director" with reasonable assurance as to information system security that can be justified and tested "in house". This could be important to provide *"best evidence"* if needed, independent of vendor statements or promises.

The idea of safe execution of trusted software is becoming critical in relationship to National Information Infrastructure Protection, and particularly in relation to e-

government. As we move about we have to acknowledge that software can't be trusted at present. We do not know the quality of such software or often where it came from. We are going to bring it "down from the Internet" and put it into our system for operational usage. Therefore we have to be sure that our system can enforce the security policy that we set and be tolerant of bad software itself. Now we can only do that with these levels of systems. We can limit potential dangers that may happen through successful penetration of the computer system. We can separate differing working environments. We can get inside and control the software development situation and isolate testing. We can clearly separate our policies from their enforcement by the computer system itself.

The underlying principle is actually quite simple. Open source licensing represents the ideal for the evaluation of the underlining security architecture in the operating system and the allied mechanisms that activate and support necessary hardware security features. At the present moment we have a problem in that we are being diverted by, I would say, by vendor attempts at "muddying the waters" achieved by trying to align system security to software quality concerns, to "bug fixes" for example. The two matters, while addressing the overall concern for information system security, are completely different.

An underlying security architecture, in the sense understood by information security professionals, means that the system itself is tolerant of software bugs and provides protection mechanisms against the effects of successful penetration. It acknowledges that they exist and that they will not "go away". Problems like "buffer overflow" will be with us forever. The simplest remedy is, as we have alluded to already, to ensure that the system security architecture will never allow the execution of data from the buffer. Hardware that already exists, for example in Intel processors, can already be used to effect this action.

The protection of the national information infrastructure and critical infrastructures of our nation will depend upon co-operation between the private and public sector. Our critical infrastructures are now in control of the private sector. That was not true 50 years ago but it is true today. Water, power, telecommunications and other vital community services could even come under the control of foreign interests and even foreign company directors. Information system security, then, takes on new significance in this environment.

In one sense, the results of such activities as SELinux are there and quite well known. Basically a trusted and acknowledged government authority in the USA, the NSA, has identified a major information security problem and the non-performance by industry in the security area. The lack of willingness to address trust requirements in computer operating systems by the IT industry has been now documented and acknowledged by the National Security Agency.

I believe government will need to upgrade security requirements for operating systems in procurement actions as a matter of urgency. Indeed, how can a manager

make any sort of statement about the security of the underlying and vital operating system in any information system and network, under IS17799, without knowing what the system is? For example, and to continue the car analogy, I don't know whether the engine on my car is going to be safe and reliable or not. I depend upon General Motors to give me that statement and I trust the statement. Managers of the future you will have a binding statement from the vendor that a system is capable of being trusted. Open source will give some reasonable basis for that evaluation. In a closed environment, such a security guarantee from a vendor has to be absolute. Reasonable prudence would thus suggest movement towards an open source solution.

I believe the evidence is there. The Sesame case and now the SELinux case illustrate that at least government has put its name down against one particular proposal, which is the viability of the open source model for assessment of security. I submit that in the age of the "next best thing" of so-called "web services", this will become even more important as we start to move to software component libraries to create bespoke application systems. We are going to have to look at the trust level of those libraries themselves.

In conclusion, I quote Professor Ed Felton of Princeton University, in a paper coming up in San Francisco in the next month. The quote concerns the right of a user to "tinker" with their system, as follows:

Freedom to Tinker: *... is the freedom to ...*

- Understand
- Discuss
- Repair, and
- Improve the technological devices you own.[159]

The simple argument against the concept that closed systems present a major advantage in relation to information security is symbolised by the machine, Collosus, which was used at Bletchley Park in the United Kingdom to break the highest level codes used by the German high command during the Second World War. The Germans felt that they had a closed system.

In the end we can but learn from history. While the encryption keys and associated data were hidden from the code breakers at Bletchley Park in the UK during WWII, they still constructed equipment and systems that enabled them to break the most secret of Germany's codes and ciphers. Advocates of a closed system could learn much from that history. In the end, depending upon secrecy of underlying technologies may give a false sense of security to the detriment of all.

[159] www.freedom-to-tinker.com

Chapter 6

The Developers' Perspective*

PAUL GAMPE
Director of Engineering, Red Hat, Asia Pacific

RHYS WEATHERLEY
Independent Developer

PAUL GAMPE

As mentioned earlier, I am the Director of Engineering for Red Hat's Asia Pacific operation. We are located in Brisbane, but we have engineers throughout the Asian region. I will talk a little bit about what we do and some of the legal issues that we face as an engineering team.

We are a team of software developers with native Asian language ability. Obviously I'm not one of the native Asian language speakers, although I do speak Japanese. We focus on Asian language text processing, input, display and printing of Asian characters. Primarily we target the Chinese, Japanese and Korean markets at this time. I am also the Red Hat representative for the Linux Internationalisation Steering Committee. We are a body that works to ensure that there is a common standard for approaching software internationalisation for the Linux platform. All of the major Linux vendors are part of that Steering Committee as well as core technology leaders from areas relevant to software internationalisation.

Our team integrates with a number of units inside of Red Hat's global engineering team. Those who are familiar with Asian text will appreciate that hieroglyphics are used instead of the much simpler character sets that are used for European based languages. For example when you wish to type one of the 26 thousand Chinese characters you have to have an input system that allows you to select that particular character. We maintain the software for each of the Chinese, Japanese and Korean languages which allows you to interact with the respective fonts and character sets for these languages.

We also work to integrate support for Asian fonts and printing into core desktop components such as GNOME and KDE. We perform language specific quality assurance locally and we also take leadership roles in emerging technologies that

* This is a revised version of the transcript from the Legal Issues for Free and Open Source Software Conference held at QUT in Brisbane Australia on 3 July 2002.

are of greater importance to Asia than other parts of the world. Two such examples are IPv6 and multi-lingual DNS, two technologies that we lead inside of Red Hat to ensure that as they develop an adoption throughout the world, Red Hat is in a position to deliver on these technologies. IPv6 is short for "Internet Protocol Version 6". IPv6 is the "next generation" protocol designed by the IETF to replace the current version Internet Protocol, IP Version 4 ("IPv4"). It is widely deployed throughout Asia as it provides a much larger address pool than the currently shrinking range of IPv4. It can also better serve the explosive growth demands for networked appliances we see in China, Korea and Japan today.

Multi-lingual or internationalised domain names allow non-English speaking people to use domain names, as we see in email addresses for example, in their native language and script.

What are some of the legal issues we face? I will speak with respect to what are some of the advantages that we are seeing and some of the legal framework that is developing throughout Asia that is assisting the work that we do in drawing on technologies throughout the region.

I will talk a little bit about the Government support in each of the countries that we are targeting including regional software compliance, the legal criteria for the sale of software throughout Asia and some of the Asia-Pacific technologies that I have mentioned earlier, such as fonts and input methods.

First, open source adoption in Asia. I do not know how many of you have Asian facing responsibilities, but Asia is the fastest growing market for adoption of open source:

> Linux adoption in the Asia-Pacific Region has grown significantly in the last year, with 1.5 percent of the region's companies having services that run on Linux, according to Gartner Inc. survey of 850 corporations in Asia.

It is not just the Linux operating system. Bill Caelli asked me to mention that the largest number of downloads of SE Linux have come from China.

What I wanted to bring forward was that Asian economies are not just adopting open source products and services, they are also beginning to embrace the open source philosophy. In 2000, India was the first Asian nation to register an affiliate branch of the Free Software Foundation in Asia.[160]

China began the process of establishing a branch of the FSF in 2000. They had completed a feasibility study by 2001 to see whether, under Chinese law, they would be able to honour the by-laws of the Free Software Foundation, and they are

[160] See: [http://gnu.org.in/]

currently about nine months into the process of registration of an affiliate branch of the Free Software Foundation in China.

So you can see that it's not just the technology they are adopting but also the fundamental understanding of what is needed to support open source software development.

There have been public statements of support from governments in all the major economies in Asia. Taiwan is backing research and development in open sourced technologies. Japan has had a multitude of initiatives supporting open source development. Korea has adopted Linux as a standard operating environment for all government organisations. The *Korea Herald*, reports that the Ministry "will establish a Linux consultative body composed of software experts from the government, academic and industry sectors to standardize Korean versions of Linux and develop a variety of programs based on the operating system". China has a long history of supporting internal development under its relationship with Red Flag Software Co. Ltd. Red Flag Software was founded by Software Research Institute of the Chinese Academy of Sciences and New Margin Venture Capital.

So why has open source been such an advantage – why are we seeing such a strong growth in open source in this region in particular?

I am not an authority on the area but I do work with developers in all these regions. These are the main reasons that I see open source is delivering an advantage to the Asian economies.

First, input methods are tightly coupled with operating systems. It is next to impossible, or it was extremely difficult, for software vendors to develop independent input methods for proprietary software operating systems. A few have emerged for the Windows platform, a few emerged for Sun, but generally if you are not part of the operating system distribution when talking about input method technology you are not going to get wide adoption.

Open source has changed that. As a technology company and, as a deliverer of open source solutions, Red Hat assesses all of the open source applications available that address our requirement for input methods. We can look at all the open source development that is occurring in Japan and China and assess which technology is going to be best to deliver on an input method for that region.

Suddenly developers are able to work and deliver technology relevant to them and can make it accessible at the operating system level. The barrier to entry has been eased.

I will talk now about the three economies for which my team develop technologies.

The Beijing Municipal Government last year (2001) requested tenders for a standard operating system and office automation environment. Of the seven tenders only one was refused.[161]

I believe the Chinese government see Linux, and Open Source in general, as a means to free itself of the burden of illegal pirated software. It is a licensing model that allows them to widely deploy software to address their business needs and not infringe on the copyright of others. We are seeing a growing adoption of the GPL for publishing software. There are a number of software initiatives occurring inside of China that are choosing the GPL licence to distribute their software. Obviously the GPL has not been contested in any Chinese Court but they acknowledge it as a good vehicle for the protection of intellectual property in their country.

Japan I think saw the benefits of open source well before many. SRA, a large software development company in Japan, were approached by the Japanese government to identify areas where they could further the adoption of Open Source software development in Japan. So the Japanese government identified this as an area for investment at least a year ago.

We have a large body of work coming out of Japan developed under the GPL licence. They are a strong exporter of open source technologies: the majority of the IPv6 reference implementation; the KAME stack and now the USAGI stack for Linux have all come out of Japan as well as the compliance testing suite, TAHI. And we are seeing government, financial, and legal support for the development of Open Source software.

I mentioned earlier the Korean Government's adoption of Linux for their standard operating environment in public service areas. One of the main areas that we are seeing of benefit for Korean companies is that they are now able to free themselves from the off-shore royalty burdens. They are able to use Linux as an operating system for their embedded technologies and no longer have to pay licensing fees to whoever the particular embedded operating system vendor may have been previously.[162]

YOPI was the first PDA to be released as a commercial device with embedded Linux and that was developed in Korea. Sharp has also just recently released a Linux based PDA [Zaurus] and they cited patents issues outside of Japan as being a reason that they chose an open source operating system.

[161] Gartner, First Take FT-15-2027.
[162] See: [http://www.embedded-linux.org/qa.php3].

In summary, what are some of the legal or legal outcomes of open sourced deployment in Asia? It is allowing these developing economies to break out of a dependency on pirated software. It is allowing them to become legally compliant and to continue to be at the forefront of technology. They are no longer victims of offshore requirements.

Open source is returning the intellectual property and technology development to the regional development communities. Surprisingly, a majority of the Chinese specific technologies that we deploy in Red Hat are based on the work of Chinese developers in China. It is lowering the barrier of entry to enabling technologies, e.g., IPV6 and multi-lingual DNS. These sorts of technologies have an open source reference implementation that is available to all. This is allowing the Asian region to get to the forefront of developing tools and software and applications for them.

RHYS WEATHERLEY

I am working on Portable.NET[163] which is a re-implementation of the program that Microsoft released a little while ago called .NET. Their material is characterised by the fact that it only runs on Windows. It does have some interesting technologies. I said: 'It would be really nice if we could run this on platforms other than Windows and make use of these technologies elsewhere'. That is the project I am working on from a developer's perspective. The team that is working on this is basically me, myself, I and Rhys, plus a few part-timers on the Net who send in a couple of lines of code each month. To date I have written about 300-odd thousand lines of code in the last year but it is still a tiny fraction of the amount of code necessary to do something of this magnitude. Because this is a very ambitious project I need help from the community. Consequently, I need to give some assurances to the community that my intentions are honourable and that I'm not going to turn out as some evil proprietary guy somewhere down the line and try to exploit the code that they contribute to me.

I decided to use the GNU General Public Licence because it creates fairness and honesty in the relationship between contributors on a project to ensure that if their contribution is born free it stays free as time goes on. Because of that, I have been able to attract a few contributors to the project and it is gradually increasing as time goes on. But if I had used some other kind of licence such as the BSD[164] there would have been two problems.

---

[163] The goal of this project is to build a suite of free software tools to build and execute .NET applications on Free Software platforms such as GNU/Linux, including a C# compiler, assembler, disassembler, and runtime engine.
[http://www.southern-storm.com.au/portable_net.html]

[164] The BSD licence allows a licensee to engage in "[r]edistribution and use in source and binary forms, with or without modification" provided the previous copyright owner is acknowledged, disclaimers are maintained in any distributed work and neither the

First of all, Microsoft or some other proprietary company without actually coming and talking to me about licensing could have just taken my code and put it in their own stuff. They could gain without giving anything back to community. That would be bad for the community. Second, there would be no incentive for the community to help me because I might go crazy and try to exploit it myself.

There are a number of other legal issues that crop up from time to time. One of them is the issue of copyright assignment. Copyright assignment can be a very big problem to deal with. I have a dual policy that people are encouraged to either assign copyright to me or to the Free Software Foundation. This comes into the registration issue – we need to have predictable registration eventually.[165] Or they can keep the copyright themselves. But the interesting thing about assignment is – what does assignment mean?[166] If someone emails some code and says "I assign this to you Rhys" is that legally binding or do they need to sign forms and stuff. Now from what Larry Rosen said earlier, select forms are necessary. This could be a big burden on open source projects. Once again, as Andrew mentioned, it involves tracking down every last person who has written some tiny little function in your code and getting them to sign a form. This is an extremely longwinded and expensive process. It is just not well suited to working in open and free software projects.

We do need a more streamlined way of trying to find how these projects can work together, have predictable copyright and copyright assignment without having these legal requirements run us into the ground with paperwork; that in reality is not really compatible with the development model that we are using. So that is a legal issue that really needs to be addressed, by legislation or by the court cases, because it is starting to bury us in legal detail that is not relevant to the production of the thing that we are producing.

Now the shared source licence.[167] I have had a bit of a personal experience with this and Andrew has as well in his own way. Recently Microsoft released this thing they call Rotor, that is their own .NET implementation which is kind of the open source you have when you are not having open source. Microsoft wants to get the effect of many eyeballs to help and fix bugs and to get contributions from the community and university lecturers to help them improve their code. But they have not yet gone the extra step that the free software and open source people talk about which is that if they want us to help them they need to help us. We need to be on an

---

organisation or the contributors of previous works are not used to endorse a derivative work. See: [http://www.opensource.org/licenses/bsd-license.php]

[165] While copyright arises automatically upon creation and fixation of a work, registration is required in the US as a precondition of filing suit for enforcement: ss 408 (a) 411 (a) *Copyright Act 1976*.

[166] For an assignment of copyright to be effective it must be in writing: s 196 *Copyright Act* 1968 (Aust), s 204 *Copyright Act 1976* (US).

[167] http://www.microsoft.com/licensing/sharedsource/default.asp.

equal footing. The core underlying principles of both the free software and the other open source licences are that all contributors are on a relatively equal footing.

My position on patents is very radical. Software patents should be eliminated, absolutely gone. There is no real justification for them to exist in the software community and part of the reason why they are so awful is because reinvention of the wheel is part of what us programmers do every single day. Every single day we come across problems that we have to solve on the fly. We write some code to solve that problem and the solutions come from our own brains or they come from things we learn in university or just experience, or even just analysing the problem in front of us and saying "OK, we can solve it that way".

The problem comes when someone runs on down to the patent office and files a patent on something that the rest of us consider just to be common sense or just an ordinary every day practice. For example, one of the guys on a related project of mine called 'DotGNU' came up with this really clever idea for downloading user interfaces. It was a clever idea. Unfortunately, I had to burst his bubble because a company called Geoworks already had a patent on it,[168] even though I think that there is probably prior art but it is really hard to find the prior art because there is just very little actually being published in this area. Most of the prior art is in code that you cannot see because most of it has been closed source in the past. And so I have had to burst the poor boy's bubble, but he had never heard of Geoworks, he had never heard this idea of doing this thing before, he completely came up with it independently.[169] The status quo is reinventing the wheel. It is just the nature of the game. If you can clearly state a problem you want solved I can solve it and usually I am going to come up with a solution just based on my own knowledge and not based of any kind of wonderful invention. So I do not think that software patents have any role.

Some people argue that there are some good patents, like patents on encryption algorithms. I argue that those patents are bad. Those patents are bad because they actually interfere with standardisation. One of the reasons why the Internet is so insecure today is because at the time that the Internet was really being built out in the 80s and early 90s we needed secure encryption systems to be built into the fundamentals of the Internet, yet they were patented. The legacy of that today is an insecure Internet. That could have been avoided if we did not have patenting of important algorithms critical to the infrastructure of computer technology. Good algorithms should be deployed as widely as possible – they should not be restricted.

[168] US patent #5,327,529.
[169] Subsequent independent creation is not a defence to patent infringement.

Chapter 7

Recent developments

GRAHAM BASSETT
Barrister, Bank of New South Wales Chambers, Brisbane, Australia

NIC SUZOR
Research Assistant, School of Law, Queensland University of Technology

OPEN SOURCE LEGISLATION

The Australian Democrats have recently proposed federal legislation which requires consideration of open source software when making decisions about public agency procurement contracts. A similar legislative proposal has been made in South Australia.[170] The Financial Management and Accountability (Anti Restrictive Software Practices) Amendment Bill 2003 (Cwth) aims to redress concerns that "a small number of software manufacturers have a disproportionate and restrictive hold on the supply, use and development of software".[171] The aim is to mandate consideration of open source software:

> An Agency must, in making a decision about the procurement of computer software for its operations, have regard to the principle that, wherever practicable, an Agency is to use open source software in preference to proprietary software.[172]

A vendor participating in a government software procurement program must ensure its software "follow industry-wide accepted standards that are open to all vendors and display an open format".[173] The data that is used in such software "will be kept at all times in a format that is completely documented in public".[174] Where agencies have purchased proprietary software it is incumbent on them in their annual report to list details of such purchases and details as to why any open source alternative was not procured.[175]

How does the Bill define open source software? It does not specifically require that they have a licensing model accepted by the Open Source Initiative. Instead,

[170] State Supply (Procurement of Software) Amendment Bill, http://www.parliament.sa.gov.au/dbsearch/lcbills_search.asp
[171] Financial Management and Accountability (Anti Restrictive Software Practices) Amendment Bill 2003, Preamble.
[172] Note 2, s44A(1).
[173] Note 2, s44A(2)(a).
[174] Note 2, s44A(2)(b).
[175] Note 2, s44A(3).

the definition asserts:

> ***open source software*** means computer software the subject of a licence granting a person a right:
>
> - without any limitation or restriction, to use the software for any purpose; and
> - without any limitation or restriction, to make copies of the software for any purpose; and
> - without any limitation or restriction, to access or modify the source code of the software for any purpose; and
> - without payment of a royalty or other fee, to distribute copies of:
> - the software (including as a component of an aggregate distribution containing computer software from several difference sources); or
> - a derived or modified form of software (whether in complied form or in the form of source code), under the same terms as the licence applying to the software.[176]

The Initiative for Software Choice (ISC) has opposed the legislation proposed by the Australian Democrats. In responding to the earlier Bill proposed in the South Australian Parliament, the group wrote a letter to the Premier, Mike Rann stating:

> The ISC strongly supports the development and adoption of all kinds of software – OSS, hybrid and proprietary. All models have a place in the highly competitive software market. Only in this manner, through vibrant and open competition, does the whole of the market thrive, and consumers – both public and private – reap tremendous benefits. Standing in stark contrast to open competition are state-mandated software preferences. These "preference" policies strip merit out of the process by using access to source code as a proxy for ICT project success ….[177]
>
> The result would be reduced options for software acquisitions, largely eliminating proprietary offerings that might be the best solutions for the given need.
>
> Additionally, constituents would suffer because the best solutions could never truly be acquired, with at least one development model – proprietary software – being restricted from agency consideration. Further, South Australia's primarily proprietary-based, ICT industry would be harmed because of foreclosed access to important state market opportunities.

The ISC group is reported as saying that such government mandates would be a barrier to free trade agreements.[178]

---

[176] Note 2, s44A(4).
[177] Letter from The Initiative for Software Choice to The Honourable Mike Rann, 10 June 2003 <http://softwarechoice.org/download_files/DearSouthAustraliaRann.pdf> at 22 September.
[178] Simon Hayes and James Riley, *The Australian IT Today*, "Open Source Trade Clash" 1 July 2003.

The proposer of the Democrats Bill, Senator Brian Greig, rebutted these claims, specifically referring to groups such as ISC. Senator Greig points out that many current government systems, often unwittingly, mandate use of proprietary systems because software procurement choices have not considered open source alternatives. The Australian Tax Office's much vaunted 'online lodgement system', Greig argues, will not work with open formats or open source software. Greig argues:

> The forces of proprietary software and their supporters have tried to portray this Bill as being protectionist in nature, one that tries to pick software favourites. It is in fact the complete opposite. Currently, we have a system that is largely based on proprietary formats, a system that does pick favourites. Removing this and opening up the playing field to all, is the raison d'etre for this Bill.[179]

Senator Greig points out that when the Thai government mandated use of open source software it was able to provide a hardware and software solution around the same price as the cost of licenses for Microsoft products alone on the same machine. The result was that Microsoft dramatically reduced its prices in order to stay competitive in the government contract area. Greig claims that Microsoft would recoup lost revenue when they provided upgrades. The key was to obtain, and then be able to control, the contract. "Microsoft's actions echo the words of Henry Ford when he offered to give away his cars provided he could keep the monopoly on spare parts. It is this type of monopoly that the use of proprietary formats maintains."[180]

SCO v IBM

In March 2003, the SCO Group (previously Caldera Systems, Inc) commenced an action against IBM in the United States District Court for the District of Utah. SCO alleges that it is the successor in title of all rights and interests in UNIX, which it derives from AT&T through a series of corporate acquisitions, and hence controls the rights of all UNIX vendors (including IBM) to use and distribute UNIX. SCO's causes of actions stem from its allegations that IBM wrongfully used code and expertise developed by SCO (and its predecessors) in developing some aspects of the Linux kernel.

In its amended complaint,[181] SCO seeks US$3 billion in damages, alleging that IBM breached the terms and conditions contained in several Software Agreements relating to Unix System V source code, by copying or adapting code into the Linux

---

[179] Senator Brian Greig, The Senate, Second Reading Speech, *Hansard*, 18 September 2003, 14672.
[180] Note 10.
[181] *The SCO Group, Inc. v International Business Machines Corporation*, Amended Complaint, 16 June 2003, <http://www.caldera.com/ibmlawsuit/amendedcomplaintjune16.html>, at 14 September 2003.

kernel, and that IBM engaged in unfair competition in aiding development of Linux. SCO also alleges that IBM misappropriated SCO's Trade Secrets, particularly the knowledge and design developed by SCO for running a UNIX-based system on Intel processors, to further development of the Linux kernel.

IBM has counterclaimed, alleging that SCO breached the terms in the Software Agreements by purporting to terminate IBM's perpetual and irrevocable UNIX rights and that SCO has publicly misrepresented the legitimacy of IBM's Linux-related products and services, in violation of the *Lanham (Trademark) Act*,[182] and that SCO infringed four of IBM's software patents. IBM also alleges that by distributing Linux products, SCO agreed under the GPL not to assert certain proprietary rights over the Linux source code, and that SCO has breached its obligations under the GPL.

The allegations have serious ramifications for the Open Source community. Most immediately, SCO has announced that it plans to charge license fees for commercial users of GNU/Linux systems.[183] If it is accepted that SCO has to power to charge license fees for existing users, we may see greater uncertainty and a slowing of the uptake of open source software in the market, by corporations not wishing to expose themselves to intellectual property obligations that are unable to be identified at the outset. Against this proposition, Eben Moglen, the Free Software Foundation's General Counsel, notes that it is impossible to assess the weight of undisclosed evidence. He agues that "a number of very severe questions arise concerning SCO's legal claims", and that he sees "substantial reason to reject SCO's assertions".[184]

More importantly, however, the claims asserted may give rise to a long awaited court interpretation of the GPL, including discussions on its classification (licence or contract), revocability, enforceability and third party liability. If litigation proceeds to completion, we can expect some very interesting precedents to be developed.

---

[182] 15 U.S.C

[183] The SCO Group, *SCO Registers UNIX® Copyrights and Offers UNIX License*, 21 July 2003, <http://ir.sco.com/ReleaseDetail.cfm?ReleaseID=114170>, at 14 September 2003.

[184] Eben Moglen, *Questioning SCO: A Hard Look at Nebulous Claims*, 1 August 2003, <http://www.fsf.org/philosophy/sco/questioning-sco.html>, at 14 September 2003.

Biographies

THE HONOURABLE PAUL LUCAS
MINISTER FOR INNOVATION AND INFORMATION ECONOMY

Paul Lucas is the Minister for Innovation and Information Economy in the Queensland State Government. Mr Lucas worked as a solicitor prior to entering Parliament, having completed a Bachelor of Economics and a Bachelor of Laws at the University of Queensland and a Master of Business Administration through the University of Southern Queensland.

The Innovation and Information Economy portfolio which Mr Lucas oversees was created to position Queensland as Australia's 'Smart State'. It seeks to identify Queensland's key strengths in innovation and emerging technologies and enable government, industry and the community to take advantage of opportunities in the new economy. Priorities for the portfolio include fostering the development of emerging knowledge industries, promoting the application of technology and e-commerce, encouraging the uptake of ICT skills and enabling bioindustries to flourish in a safe and ethical environment. Mr Lucas sees the new portfolio as helping build on Queensland's existing natural, economic and human resource advantages.

PROFESSOR BRIAN FITZGERALD BA (Griff), LLB(Hons) (QUT), BCL (Oxon), LLM (Harv) PhD (Griff)

Brian Fitzgerald is Head of the School of Law at Queensland University of Technology, Brisbane, Australia and holds postgraduate law degrees from Oxford University and Harvard University. He is co-editor of one of Australia's leading texts on E Commerce, Software and the Internet – *Going Digital 2000* - and has published articles on Law and the Internet, Technology Law and Intellectual Property Law in Australia, the United States, Europe, India, Nepal and Japan. His latest publication (with his sister Anne Fitzgerald) is *Cyberlaw: Cases and Materials on the Internet, Digital Intellectual Property and Electronic Commerce* (2002). Over the past three years Brian has delivered seminars on information technology and intellectual property law in Australia, New Zealand, USA, Canada, Norway, India, Nepal, Japan and the Netherlands. In October 1999 Brian delivered the Seventh Annual Tenzer Lecture – Software as Discourse: The Power of Intellectual Property in Digital Architecture – at Cardozo Law School, Yeshiva University in New York. In October 2000 he was invited as a part of the Distinguished Speaker series hosted by the Ontario wide Centre for Innovation Law and Policy to deliver an address on Digital Property at the University of Western Ontario Law School in London, Canada. During the first half of 2001 he was a Visiting Professor at Santa Clara University Law School in Silicon Valley in the United States, teaching a seminar on Digital Property www.scu.edu/law/FacWebPage/Fitzgerald. In March 2001 he convened a forum on

"Innovation, Software, and Reverse Engineering: Technological and Legal Issues" and in June 2001 he organised a seminar on "Legal and Business Issues Relating to Open Source Software" both held at Santa Clara University in Silicon Valley. From 1998-2001 Brian was Head of the School of Law and Justice at Southern Cross University in NSW.

MARK WEBBINK

Mark Webbink joined Red Hat in 2000 having previously worked at the law firm of Moore & Van Allen, where he practised in the field of intellectual property transactions. Since joining Red Hat Mark has spoken at numerous conferences around the world on the subject of Open Source. Mark has addressed the Licensing Executive Society and the Practising Law Institute. He has also testified before the United States Congress on Open Source and Patent Law as well as the United States Federal Trade Commission and the United States Justice Department on Patent Law and Competition. Mark holds a law degree with high honours from North Carolina Central University. He has a Masters degree in public administration from the University of North Carolina and a Bachelor of Arts degree from Purdue University.

PROFESSOR WILLIAM J (BILL) CAELLI — FACS, FTICA, MIEEE (SEN)

Bill Caelli is the Head of the School of Software Engineering and Data Communications. Immediately prior to this he was Head of the School of Data Communications (1994) which merged with the then School of Computing Science and Software Engineering on 1 March 2002. Prior to this he was the Founding Director (1988) of the Information Security Research Centre (ISRC), at the Queensland University of Technology (QUT). He was a Founder of ERACOM Pty. Ltd., a major information technology security company with global affiliations and served as its Managing Director and then Technical Director from its commencement in 1979 until mid-1998. Bill was made a Member of the Australian Science Technology and Engineering Council (ASTEC) in August 1995 and served on that Council till its operations were merged with the pertinent Prime Minister's Council in mid-1998.

PETER JAMES

Peter James specialises in information technology, ecommerce, telecommunications and trade practices law and is a partner in Communications, Media and Technology Group for Allens Arthur Robinson, Sydney. His commitment to finding commercially viable solutions, and providing personal attention, fast turnaround and cutting-edge legal advice, has built his reputation as one of Australia's leading technology lawyers. He had advised governments in relation to telecommunications regulation and infrastructure projects. Peter also acts for suppliers to the telecommunications industry including Nortel Networks.

PAUL GAMPE

As Director of Engineering for Red Hat Asia-Pacific, Paul Gampe manages multiple development teams to internationalise products and services for Red Hat. Paul also manages the engineering team that, over the last few years, has added Japanese, Korean and most recently Chinese language support to core Red Hat products. Prior to Red Hat, Gampe served as technical operations manager for Asia Pacific Network Information Centre (APNIC), one of the three worldwide registries that govern the distribution of Internet resources (such as IP addresses). As technical operations manager, Gampe over saw the construction of an Asia-wide registry infrastructure. Gampe joined APNIC in 1997 after serving as General Manager of TWICS, the first public ISP based in Japan. Gampe is a member of the Linux Internationalization Steering Committee and Chair of the special interest group on DNS for APNIC.

RHYS WEATHERLEY

Rhys Weatherley is a member of the DotGNU Steering Committee, and the primary author of Portable.NET. He graduated from The University of Queensland in 1990 and has since worked at various Australian and US companies. He recently returned to Australia to establish his company, Southern Storm Software, Pty Ltd.

BILL LARD

Bill Lard is Senior Director of Licensing Strategy & Architecture at Sun Microsystems, Inc. He has been an Attorney with Sun for nine years handling software related matters. His current role is to establish the future direction of Sun's overall technology licensing strategy and architecture.

YANCY LIND

Yancy is CEO of Lutris Technologies in Santa Cruz, an Internet middleware software company. He is a businessman, not an attorney, and aims to make open source companies commercially successful.

DAVID SCHELLHASE

David Schellhase, works in-house as an attorney with Linuxcare, a Linux services company. He is currently writing a book *Inhouse: The Practise of Law Inside an Emerging Growth Company*. He has also worked as an attorney for a number of law firms in Silicon Valley.

LARRY ROSEN

Lawrence E. (Larry) Rosen is an attorney and founding partner of Rosenlaw.com, a law firm in California. He is a computer specialist and has extensive experience teaching computer programming. Larry has been a department and product manager in the computer and communications industry. As an attorney, his specialty is technology, but he is also a skilled litigator and negotiator, and a legal advisor to individuals and companies throughout the Bay Area and the world. He is executive director of Open Source Initiative, a non-profit organization that reviews and approves open source licenses and that manages the "OSI Certified" certification mark for open source software.

GRAHAM BASSETT — BA, Dip Ed, InfoTech, LLB(Hons), Barrister-at-law

Graham Bassett is a barrister of the Supreme Court of Queensland and the High Court of Australia with a special interest in intellectual property and information technology. He has Chambers in Brisbane and maintains an office in Byron Bay. Before coming to the Bar he developed IT systems, mainly in private schools in Sydney, Australia. He lectures in Information Technology Law and Intellectual Property.

NIC SUZOR

Nic Suzor is a final year student in the Information Technology and Law faculties at Queensland University of Technology. He has been working as a programmer since 1997 and has been active in several open source development projects.