
**SEEING THE MATRIX:
What Neo Could Tell Us About the Value of Open Code
in the Age of Thinking Machines
or
The Licensing Model and Implications of Open Code Software**

PAUL ADAMS

**A dissertation presented in fulfilment of the requirements of the
University of Auckland for the degree of Bachelor of Laws (Hons)**

**Auckland
February 2001**

© Paul Adams, 2001

This work is the copyright of the author. The author asserts his moral rights and right of identification under the Copyright Act 1994. It may be reproduced in full or part provided that authorship is acknowledged and this copyright notice is included intact.

This paper examines a new form of software: open code software. In the last three years open code software has risen to become a serious threat to the commercial software industry. Open code software differs from traditional software in that the user has access to the software's 'source code' - the underlying code that software developers read, edit and compile.

The purpose of this paper is to undertake a detailed legal analysis of three key issues surrounding open code software. Firstly, to examine open code software's unique licensing model, known as 'copyleft', that has been instrumental in the proliferation and development of open code software. Secondly, to analyse the implications that open code software may have for common law intellectual property regimes, in particular its role in restoring 'balance' to intellectual property law. Thirdly, to consider open code software's unique ownership regime and the implications this may have for the ability of democratic governments to effectively regulate open code software.

About the Author

Paul is Accelerator Manager of The Icehouse, a partnership between nine of New Zealand's leading business organisations to promote the development of high growth ventures in New Zealand. As Accelerator Manager Paul is responsible for assisting young, high potential technology ventures to achieve rapid growth and maximise shareholder value. See www.theicehouse.co.nz

Paul has worked with and consulted to a number of technology ventures in New Zealand and the United States. In 2000 Paul co-founded Cytrinity Inc., a US based Information Services start-up that was sold to Southeast Asian interests in 2001. Paul holds a Law (Honours) Degree in Information Technology Law and a Bachelor of Arts Degree from The University of Auckland. He can be reached at byron@ihug.co.nz

There is a scene in the 'The Matrix' in which the hero, Neo, played by Keanu Reeves, faces his nemesis, Agent Smith. Trapped inside a hotel corridor that is actually a computer generated virtual reality, Neo is defenceless. That is, until he becomes aware of the 'virtualness' of the reality around him and sees for the first time the actual computer code that constructs this reality. Seeing this reality as code, he is free to change it as he desires and so he repeals various physical laws, destroys his opponents and gets the girl.

Now for a film that doesn't pretend to be much more than it is, an entertaining and action packed Hollywood blockbuster, one really wouldn't expect much in the way of high brow commentary. Except that, at risk of being accused of the adage a 'blind man sees what he wants to', Neo accidentally or not, has pointed out something quite important. That is, in a world increasingly dependent on thinking machines, machines that run on computer code, there is significant value in that code being 'open', in being able to see what that code is and what rules or laws it imposes on those who can't see it. And it is this point, intentioned or not, that will occupy this paper.

In the last three years a new form of software has risen to prominence in the information technology industry. It is unique in a number of ways: its technical structure, its licensing model, its development model. It is also unique in that unlike virtually every other form of software that has gone before, it is not only surviving, but thriving against the industry's dominant paradigm - closed commercial software. This software is open code software.

The purpose of this paper is to examine the open code software phenomenon. Much has been written focussing on the technical aspects of open code software, but to date little attempt has been made to undertake a thorough legal analysis of the new software or its unique development model. Though this paper could hardly pretend to do justice to such an analysis it will attempt to focus on three key issues concerning open code software. Firstly, it will examine the unique licensing model under which open code software is distributed and compare it with the licensing models of the other major forms of software. Secondly, it will discuss open code software's potent implications for intellectual property law. Thirdly, it will consider the potential impact that such software may have on regulation generally and the state's ability to govern in the 21st century. Finally, as an ancillary goal, this paper hopes to provide the non-technical reader with a general understanding of what open code software is and how it functions.

In exploring these three issues this paper will adopt a bipartite structure. Part I asks the question: what is open code? The aim in Part I is to clearly establish the nature of open code software to enable subsequent detailed analysis. Part I divides into two sections: Section A briefly explains the technical distinction between 'open code' and 'closed code' software, keeping in mind that many readers of this paper may not possess the technical background that is necessary to inform many of the concepts at work here. Section B examines the legal distinction between open code software and the industry's other dominant models. Section B has three subsections. Subsection 1 outlines the theoretical foundations of the intellectual property models that define the different software types or 'models'. Subsection 2 applies this information to the models themselves, developing a taxonomy of the industry's standard models, before describing the licensing features of each in turn. Part I concludes with subsection 3, which undertakes a detailed analysis of the key provisions of the main open code software licensing model, the GNU Public License. In doing so, subsection 3 addresses the first of the three issues discussed above that this paper hopes to touch upon.

Part II will form the bulk of this paper. As with Part I, it asks a question of us: why should we care about open code software? Part II's purpose is to reveal to the reader its importance and implications. It is divided into two sections. Section A addresses the technical aspect of open code software. It begins by discussing the importance of the open code

development model before splitting into three subsections. Subsection 1 details the history of the open code movement and the current market position of open code software. Subsection 2 analyses the open code development model, first discussing traditional software development before turning to examine in detail the open code model itself. Subsection 3 analyses the software produced by this model and the remarkable efficiencies it generates. Section B occupies the bulk of Part II. It analyses open code software's profound implications for intellectual property law and democratic governance. It has two subsections. Subsection 1 considers the theoretical foundations of intellectual property law and the potential negative consequences of modern reforms. It then turns to consider the role open code software may play in restoring balance to modern intellectual property regimes. The paper concludes with Subsection 2 which analyses open code software's unique ownership regime and the significant impact this has on a government's ability to effectively regulate such software. It is hoped that this structure will clearly elucidate the three key issues discussed above and reveal to the reader the unique features of open code software.

A Word About Terminology

The terminology surrounding open code software is at times a source of considerable confusion. Open code software is generally known as 'open source software' or less commonly as 'free software' [here, free as in 'freedom']. Each name is promoted by its own advocacy group. To avoid giving preference to either group, this note will adopt the expression 'open code software' to refer the phenomenon generally, unless one or other group is specifically intended.

PART I: WHAT IS OPEN CODE?

'Open code' is a multifaceted concept. It has been variously described as a technical state, an innovative licensing model, a software movement, a developmental process, and lastly as a new philosophy of intellectual property. Accordingly, answering the question 'what is open code?' is no small feat. Probably the simplest thing one can say about 'open code' is that it is computer software. With this in mind, we begin with the basics, by looking at two key features of open code software:

- the technical distinction between 'open' and 'closed' source code
- the intellectual property and licensing models under which software is distributed.

With neither feature is it appropriate to restrict our analysis solely to open code software. Rather we will proceed in each case by comparative analysis between the industry standard and open code software. In the case of the technical distinction, the comparison is a simple one between 'open' and 'closed' source code. In the second case the comparison is more complicated, requiring that we first examine the theoretical foundations of software licensing agreements, before constructing a general taxonomy of software. With this taxonomy in place we will investigate each software model in turn, focussing our attention principally on the distinction between the industry standard, the commercial software licensing model, and the licensing model of open code software.

SECTION A: THE TECHNICAL DISTINCTION¹

Open code software is software for which both the *machine code* [binary file] and the *programme source code* is freely accessible to both developers and users.²

To those without a technical background this definition does not advance our understanding overly. We first need to understand two related concepts: 'source code' and 'machine code'.

What is source code?

Program instructions in their original form. The word 'source' differentiates code from various other forms that it can have (for example, object code and executable code). Initially, a programmer writes a program in a particular programming language. This form of the program is called the source program, or more generically, source code.

To execute the program, however, the programmer must translate it into machine language [binary format], the language that the computer understands. Source code is the only format that is readable by humans. Generally, when you purchase a program, you receive it in its machine language [binary] format. This means that you can execute it directly, but you cannot read or modify it.³

What is machine code?

¹ If the reader is familiar with the concepts of 'closed source code' and 'open source code' then he or she can skip this section and proceed directly to Part I, Section B.

² 'open source software', (definition). Computer User.com High Tech Dictionary, <http://www.computeruser.com/resources/dictionary/>, last visited: 11/01/01.

³ 'source code' (definition) Webopedia: Online Computer Dictionary, http://webopedia.internet.com/TERM/s/source_code.html, last visited: 11/01/01.

The representation of a computer program which is actually read and interpreted by the computer. Machine code is stored in binary format. A binary file is computer-readable but not human-readable. All executable programs are stored in binary files, as are most numeric data files.

Most commercial software is distributed to users in binary format without the software's source code. Accordingly, it is extremely difficult to read or modify the software.⁴

The following provides a graphic example of the key difference between source code and machine code. Below is an example of a programme designed to calculate the distance between two points in a plane.

Source Code:

```
float
distance (p0, p1)
    struct point p0, p1;
{
    float xdist = p1.x - p0.x;
    float ydist = p1.y - p0.y;
    return sqrt (xdist * xdist + ydist * ydist);
}
```

Machine Code:

This is the same program in machine code or in executable form [running] on a computer:

1314258944	-232267772	-231844864	1634862
1411907592	-231844736	2159150	1420296208
-234880989	-234879837	-234879966	-232295424
1644167167	-3214848	1090581031	1962942495
572518958	-803143692	1314803317 ⁵	

This example demonstrates the pivotal role source code plays in software development. While possession of the machine code [binary format] enables an individual to run a programme, without access to the programme's source code it is virtually impossible to read or modify the program.⁶

It is this situation that commercial software developers exploit to produce and distribute software. Software is written as source code by developers then shipped to the user as machine code or 'binaries'. The user can run the programme but cannot read the source code to see how it works nor modify it to improve its performance. By way of analogy, it is as if the user has purchased a new car and has driven it off the lot but is unable to open its hood to see how it works or fix a broken connection.

This brings us to the key feature of open code that will occupy us for the remainder of this paper and save us from a purely technical discussion of software programming. If we return to our definition:

⁴ 'machine code' (definition). Free Online Dictionary of Computing, <http://foldoc.doc.ic.ac.uk/foldoc/foldoc.cgi?machine+language>, last visited: 11/01/01; 'binary file' (definition) Webopedia: Online Computer Dictionary, http://webopedia.internet.com/TERM/b/binary_file.html; last visited: 11/01/01.

⁵ Richard Stallman, 'Why Software Should be Free', The GNU Project, <http://www.fsf.org/philosophy/shouldbefree.html>, last visited: 05/12/00.

⁶ Highly proficient programmers are able to reconstruct source code from machine code but the process is extremely complex and laborious. Given that a program may contain millions of lines of code, as a practical matter reverse engineering is impossible.

Open code software is software for which both the *machine code* and the *programme source code* is freely accessible to both developers and users.⁷

This means with the open code software not only can the user *run* the software, but they can *read* and *modify* it as well. From a practical perspective this means the user has access to both the binary *and* the source code versions of the software. This is in sharp contrast to *closed* source code software, where the source code is *not* available and consequently the user is restricted solely to running the software. Most software produced in the world is closed code software.

This technical distinction between software for which the source code is available [open source], versus software for which the source code is unavailable [closed source], is central to the open code debate in that it creates a very sharp distinction in the software industry. Software is either distributed with, or without, source code - there is no middle ground. However, it does not in itself explain the essential character of 'open code' software. Equally important is the intellectual property model under which the software is licensed and distributed, and it is to this element that we now turn.

SECTION B: THE INTELLECTUAL PROPERTY DISTINCTION

Subsection 1: The Basics of Software Intellectual Property Models

Software is commonly defined by reference to its intellectual property model. This model catalogues certain key features and rights associated with that software, including but not limited to the following characteristics:

- is the software owned?
- if so, by whom?
- how does the owner control it?
- what rights does this control impart?
- under what terms is the software distributed?

There are a variety of intellectual property models common to software development. These range from pure commercial software to public domain software. Each model affects the manner in which the software may be distributed and used. In the vast bulk of cases software is 'owned' in that it has an identifiable 'owner' who holds the intellectual property rights to the code that underlies the software.⁸ Usually this owner is a single individual or organisation although sometimes a number of individuals or organisations may jointly hold these rights. In certain cases software may have no owner, the software effectively exists in the commons; public domain software is one such example.

In the majority of cases, control or ownership of software stems from copyright.⁹ In the United States for example, copyright automatically attaches to software upon its creation.¹⁰ Hence, copyright is the basis of most software intellectual property models.

Copyright vests in the copyright holder certain rights in relation to the copyrighted software; these include:

- the right to copy the work
- the right to distribute the work

⁷ Supra note 2.

⁸ A software program, application or utility cannot in itself be copyrighted - it is the underlying source code that is copyrighted. A programme may however in certain circumstances be patented.

⁹ Less commonly control may derive from patent.

- the right to prepare derivative or modified versions of the work.

The holder of these rights can prevent others from exercising them or may make their exercise subject to certain conditions (such as the payment of a fee, or agreement to certain terms). The holder can also assign the rights or forgo them. Copyright also vests certain rights in the users of the copyrighted work. The most important of these is the right to 'fair use'. Fair use is an amorphous concept that, in basic terms, provides that certain uses of copyrighted works, called 'fair uses', are not an infringement of copyright, even though they may involve copying, adapting or displaying the work. If a particular use is a 'fair use' the owner has no right to demand that the use be first authorised or subject to any conditions imposed by him or her.¹¹

In the case of digital media such as software, copyright holders are frequently reluctant to distribute software subject only to rights derived from copyright because of the fair use doctrine. If they did so, users would be able to take advantage of its broad exceptions to liberally use, distribute and resell the software. Since the early 1980's software developers have avoided this problem by using software licences. A software license is a contract between the developer and the purchaser of a piece of software that carefully circumscribes the rights of the purchaser in relation to that software. As private law between consenting parties, licensing agreements supersede default laws governing the distribution of intellectual property rights, although this is by no means settled law.¹²

The almost universal adoption of licensing agreements by software developers has meant that software licences are now the preeminent legal tool for defining rights in relation to software. Software developers copyright software then use copyright holder rights to generate licensing agreements that tailor the user's rights to the level the developer desires. As a result, software licences are now the key influence on the intellectual property model of any given piece of software, because it is the licence (rather than the underlying copyright) that defines the rights catalogued in the software's intellectual property model. As a consequence, it is software licences that are the key factor in delineating different types of software from one another.

As a final note, the technical distinction between open and closed source code discussed above is an important but not fundamental factor in determining the intellectual property model that governs the software. Generally speaking, closed

¹⁰ T. Stanco, 'Software Licenses and traditional Copyright Law', Linux Programming, [wysiwyg://87/http://www.linuxprogramming.com/story.php3?ltsn=2000-07-001-03-ID-UU](http://www.linuxprogramming.com/story.php3?ltsn=2000-07-001-03-ID-UU); last visited 05/12/00.

¹¹ A brief discussion of fair use follows: excerpted from Judge Whyte's decision, September 22, 1995 in *RTC & Bridge v Erlich, Netcom & Klemesrud*. [note: this is the United States legal interpretation of fair use].

'Infringement' consists of violating the author's exclusive rights. 17 U.S.C. Section 501. Although the author has the exclusive rights to reproduce, distribute, and display a copyrighted work under section 106, these rights are limited by the defence of "fair use." Notwithstanding the provision of Section 106A, the *fair use of a copyrighted work,* including such use by reproduction in copies... or by any other means specified in that section, for purposes such as criticism, comment, news reporting, teaching (including multiple copies for classroom use), scholarship, or research, *is not an infringement* of copyright. 17 U.S.C. section 107. The defence "permits and requires courts to avoid rigid application of the copyright statute when, on occasion, it would stifle the very creativity which that law is designed to foster." *Campbell v. Acuff-Rose Music, Inc.*, 114 S.Ct. 1164, 1170 (1994) Congress has set out four nonexclusive factors to be considered in determining the availability of the fair use defence:

- (1) the purpose and character of the use, including whether such use is of a commercial nature or is for non-profit educational purposes;
- (2) the nature of the copyrighted work;
- (3) the amount and substantiality of the portion used in relation to the copyrighted work as a whole; and
- (4) the effect of the use upon the potential market for or value of the copyrighted work.

17 U.S.C. section 107. The fair use doctrine calls for a case-by-case analysis. *Campbell*, 114 S.Ct. at 1170. All of the factors "are to be explored, and the results weighed together, in light of the purposes of copyright." *Id.* at 1170-71.

¹² See generally: Elkin-Koren N., 'Copyrights in Cyberspace - Rights without Laws?', *Symposium on Internet and Legal Theory, Chicago-Kent Law Review*, v. 73, n. 4, 1998, pp. 1155 - 1202; & Fisher W., 'Property and Contract on the Internet', *Symposium on Internet and Legal Theory, Chicago-Kent Law Review*, v. 73, n. 4, 1998, pp. 1203 - 1257.

source code restricts the range of intellectual property models available to the developer, while open source code enables greater choice. This is due to the fact that generally developers who distribute closed code software are profit motivated. Consequently, they are forced to employ a relatively narrow, standard set of restrictive licensing conditions, to ensure that the software is sufficiently protected to enable a return on the intellectual property assets it represents. Conversely, open code developers are less frequently profit motivated¹³ and consequently are largely free to choose from a wide range of licensing terms, as they need be less concerned to ensure complete protection of the software. The exception to this rule is open code software licensed under the Gnu Public License, whose terms, while promoting user freedom are set out in a standard license. The Gnu Public License is discussed in considerable detail below; see Part I, Section B, Subsection 3.

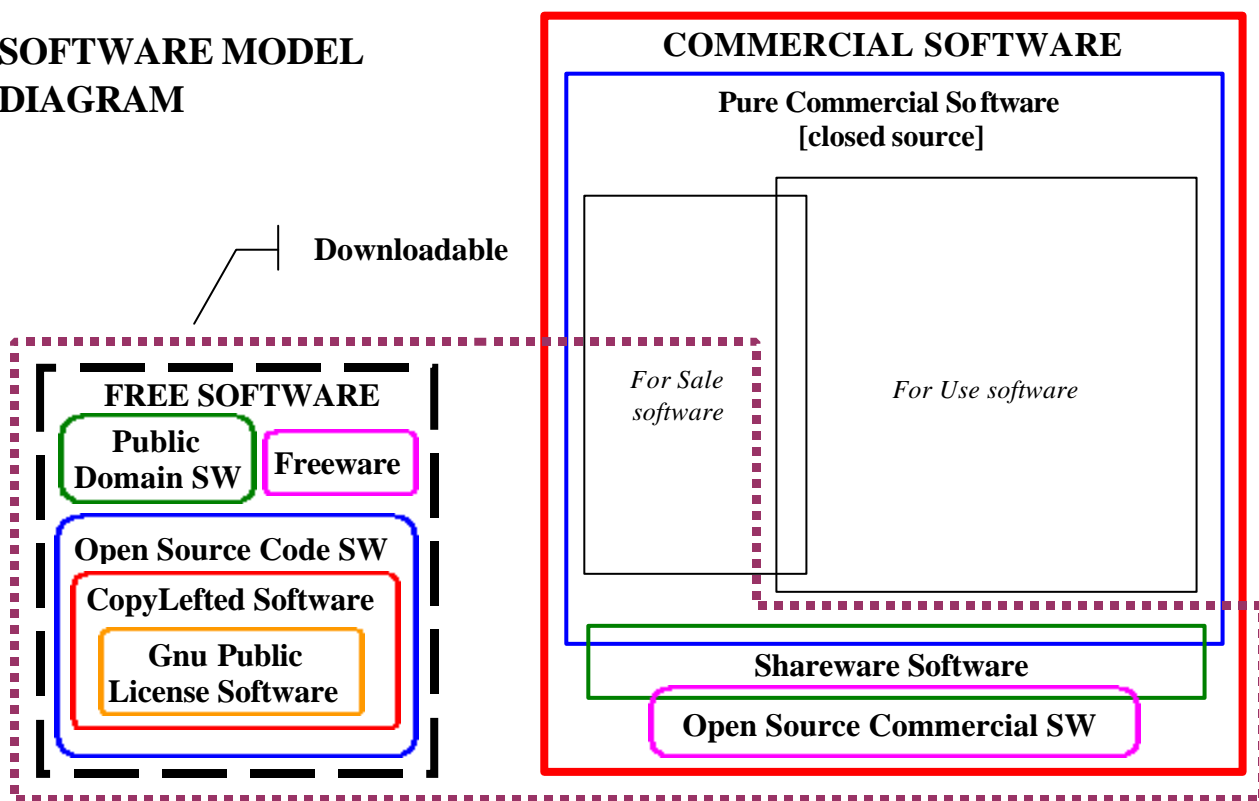
Subsection 2: The Different Software Models

Having clearly outlined the theoretical foundations of software intellectual property models, we can now turn to a full discussion of the different types of software and the intellectual property models that overlay them. Subsequent discussion will concentrate on two main models:

- the software industry’s dominant model - pure commercial software; and
- the model that is the subject of this note - open code software

The following diagram illustrates the different types of software as defined by their intellectual property models. The size of the boxes relative to each other is intended to provide a rough guide to the predominance of that software model within the industry. The dotted line which surrounds the Free Software box and cuts through the Commercial Software box encompasses those forms of software that are available to users via download from the Internet.

SOFTWARE MODEL DIAGRAM



Each category represented in the diagram is discussed below.

Commercial Software

¹³ This is partially due to the fact that it is more difficult to profit from open code software using traditional business models.

The box on the right hand side of the diagram represents commercial software. Commercial software is software developed for profit and or commercial enterprise. Within the broader commercial software group are three subgroups:

- pure commercial software
- shareware software
- open code commercial software.

Pure commercial software is the dominant commercial model although the success of open code software generally has seen an increase in the popularity of open code commercial software. Shareware remains a tried and tested, though not especially successful software model.

Pure Commercial Software

The largest box within the greater commercial software model represents pure commercial software. As the name implies, pure commercial software is the most commercially orientated of all software models and therefore tends to be subject to the most restrictive licensing terms. There are two basic sub-types:

- For Use - software that has been developed either in-house or on order to perform specific tasks for a specific organisation. The low public profile of For Use software disguises the fact that it composes the bulk of all commercial software developed. A database written internally by a book wholesaler is an example of For Use software.
- For Sale - software that has been developed for the purpose of sale. For Sale software is better known than For Use software because it is distributed to the public. Most substantive software written for desktop computers i.e. Microsoft Windows or ClarisWorks is commercial For Sale software.

For Use software is rarely distributed, so the issue of source code availability generally does not arise. In the rare circumstances in which it is distributed, it is usually as closed code. Likewise, For Sale software is typically distributed as closed code software to protect the development investment made.¹⁴ This means the software is distributed to users in binary form only, which ensures that it can be run but not modified or reversed engineered. Pure commercial software is almost inevitably subject to licensing restrictions governing its use, redistribution and modification. The details of these licenses vary, but a number of common threads can be identified. These are explored below.

Commercial Software License Terms

Use Rights

Commercial licences tend to restrict use solely to the personal or commercial use of the individual or organisation who purchased the software. Typically a single legal copy of the software will be required for each workstation the software is to be used at.¹⁵ Alternatively restrictions may be made on the number of users¹⁶ or the number of simultaneous users at any one time.¹⁷

Copying Rights

¹⁴ Distributing the source code of 'For Sale' software to end users effectively abdicates any control over the software and consequently any profits. Distributing the source code of 'For Use' software enables competitors to build from your development costs.

¹⁵ Known as 'per-copy licensing' see: P. Johnson, 'Liberal Source Software', Liberal Source Homepage, <http://www.elj.com/lss/lss.html>; last visited: 23/12/01.

¹⁶ Known as 'seat licensing'. Ibid.

¹⁷ Known as 'concurrent user charges'. Ibid.

Virtually all commercial software licenses prohibit the user from making copies of the software except for the purposes of making a single archival or backup copy.¹⁸

Modification Rights

Although the fair use doctrine provides users with a fairly broad entitlement to modify original works, commercial software licences tend to rescind such user rights. Typically, users are prohibited from correcting, modifying and or enhancing licensed software in any way.¹⁹ This prevents the evolution of the software except at the initiative of the developer.

Distribution Rights

Default copyright regimes grant the user the right to a 'first sale' of legally purchased copyrighted works. Commercial licences typically prohibit any redistribution or resale of software.²⁰ In practice this right is extremely hard to enforce against individuals, although less so against corporations.²¹

Warranties

Most software licences disclaim warranties of performance to avoid liability for software defects and bugs. Many licences also disclaim warranties of merchantability and fitness.²²

Taken together, the restrictive terms of commercial software licenses act to augment the intellectual property rights and enhance the profitability of software developers. However, such benefits occur at the expense of the user, who bears the cost in terms of loss of freedom to use and enhance software and increased purchase prices caused by the market power of the major developers. Drawing back from the detail of the commercial licensing model it is clear that such licenses are carefully constructed to afford the greatest possible protection and benefit to large corporate developers. This balance between developer rights and user rights is a recurring theme in the software industry. The commercial software model establishes one pole of a dialectic whose opposite end terminates in the open code model. This dialectic between user and developer rights is the subject of considerable discussion in Part I, Section B, Subsection 1 of this paper.

Shareware

A variant of the commercial model is shareware software. Shareware products are fully functional and freely redistributable but have a licence that mandates eventual purchase by both individuals and corporations.²³ Developers depend largely on the honesty of those who choose to keep the software to pay the stipulated licence fee.

Open Code Commercial Software

Encouraged by the success of the open code movement, a small number of commercial developers have moved to release the source code of their software. The motivations for such moves vary but have generally focussed on achieving a 'Loss-

¹⁸ Supra note 10.

¹⁹ Ibid.

²⁰ Ibid.

²¹ Corporations face severe penalties for software piracy and are much more vulnerable than individuals to being betrayed by disgruntled employees or customers to authorities.

²² Supra note 10.

Leader'²⁴ position within markets that tend to be dominated by powerful network economic effects.²⁵ The basic idea is as follows: releasing software source code tends to speed the proliferation and adoption of that software. The more individuals who use the software, the more valuable the software becomes to each user²⁶ and hence the more likely it is to attract prospective users. This creates a virtuous circle that eventually results in the developer acquiring a dominant position within the market. Having acquired this position the developer can then use it to generate revenues via other means.

The pre-eminent example of open code commercial software, and correspondingly this strategy in action, is Netscape Corporation's decision to release the Mozilla source code to the Netscape Communicator Browser in 1998.²⁷ Netscape executives cited a seminal open code work, "The Cathedral and the Bazaar" by Eric Raymond, as a key factor in their decision.²⁸ Netscape's move is widely regarded as having been crucial to its successful defence of the Netscape Browser against aggressive moves by Microsoft's Internet Explorer to establish market dominance.²⁹ It is also credited with heading off Microsoft's attempt to dominate HTML³⁰ by creating a proprietary standard in Internet Explorer.

Despite this success, few major developers have undertaken substantial open code commercial software initiatives, although the numbers are slowly increasing. Ironically, in the Halloween Documents, Microsoft, often painted as the arch-enemy of the open code movement, explicitly discusses releasing the source code of some its products to combat the movement and retain its market share.³¹

Free Software

The broken-lined box on the left-hand side of the diagram represents Free software. 'Free' in this context refers to 'freedom' rather than price. Specifically, the high degree of user freedom (and conversely the lower degree of developer control) associated with these forms of software. There are three basic types of Free software:

- public domain software
- freeware software
- open code software

²³ V. Valloppillil, 'Open Source Software - A (New?) Development Methodology', a.k.a 'The Halloween Documents', I, v1.14, Executive Summary, <http://www.opensource.org/halloween/halloween1.html>, last visited: 23/12/00.

²⁴ E. Raymond, 'The Magic Cauldron', Eric Raymond Homepage, June 1999, <http://www.tuxedo.org/~esr/writings/magic-cauldron/>; last visited: 04/02/01.

²⁵ Also known as "increasing returns on production", a network economic effect occurs in a situation where a product grows more valuable both to each individual consumer and overall as more consumers use that product. Network economic effects tend to encourage the creation of monopolies within information industries. For a general discussion of network economic effects see: M. Lemley & D. McGowan, Legal Implications of Network Economic Effects, 86 California Law Review, 479 (1998).

²⁶ The case of Microsoft Word is instructive: if you wish to exchange Word documents with another user, both parties need to be using Word to effect a successful transfer. Therefore the greater the number of people that use Word the greater the number of people you can exchange documents with, increasing the utility of the programme not simply to you but to every other user. Note: it is possible to make an exchange across programmes and even platforms (i.e. PC to Macintosh) but the process is more complicated and time consuming.

²⁷ Supra note 24. For a full discussion of the Netscape's decision see: J. Hamerly, T. Paquin & S. Walton, 'Freeing the Source: The Story of Mozilla', Open Sources: Voices from the Open Source Revolution, O'Reilly Publishing, 1999.

²⁸ E. Raymond, The Cathedral and the Bazaar, Eric Raymond's Homepage, <http://www.tuxedo.org/~esr/writings/cathedral-bazaar/>; last visited: 04/02/01.

²⁹ Including shady bundling practices that would later become the centrepiece of the Department of Justice's antitrust suit against Microsoft.

³⁰ 'Hyper Text Mark Up Language' (definition) - the authoring language used to create documents on the World Wide Web. HTML defines the structure and layout of a Web document; <http://webopedia.internet.com/TERM/H/HTML.html>; last visited: 24/01/01.

³¹ Supra note 23.

Both freeware and public domain software are relatively rare. Since the early 1990's open code software has undergone a renaissance and is rapidly growing in popularity. All three forms are discussed below:

Public Domain software

Public domain software is software that is not copyrighted. Accordingly, it may be used, distributed and modified without restriction. However, because it is not copyrighted, it can be adapted by developers to create new software that *is* subsequently copyrighted, removing the derivative (but not the original) from the public domain. Many of the Internet's coordinating standards including TCP/IP and HTML are examples of public domain software. Public domain software may or may not be distributed with its attendant source code.

Freeware software

The term 'freeware' is commonly used to describe software for which redistribution but not modification is possible. In most cases, to prevent modification the software source code is not distributed.

Open Code software

Open code software is software that demonstrates the following characteristics:

- the software's source code is freely available
- users of the software enjoy the following freedoms: [collectively known as the 'central freedoms']
 1. The freedom to run the program, for any purpose
 2. The freedom to study how the program works and modify it
 3. The freedom to redistribute copies
 4. The freedom to improve the program, and release the improvements to the public.³²

The amalgamation of freely available source code with the central freedoms provides the user with a degree of liberty significantly in excess of other forms of software. Open code software enables the user to run, copy, distribute, study, and modify that software without restriction. This is in sharp contrast to the intellectual property model under which most software is distributed, the pure commercial software model, which restricts the user solely to running the software and making a single archival copy.

The freedoms associated with open code software operate without restriction - the user is not required to ask for permission nor pay a fee to exercise any of the freedoms or to access the source code. Instead the freedoms operate as an innate function of the user's possession of the software. However, open code software may be sold, in that any holder of such software may, at his or her discretion, choose to charge a fee to distribute the software to others. If a fee is payable, it may be set at any price the distributor desires. Regardless of whether a fee is actually paid, any recipients of the software must have access to the source code and continue to enjoy the benefits of the central freedoms. This means that even if a user purchases open code software she is still free to redistribute it, either gratis or for a fee.

Open code software organisations actively encourage individuals to sell open code software,³³ but why would anyone voluntarily choose to pay for free software? Sale fees may be justified a number of ways:

³² 'What is Free Software?', The Free Software Foundation, <http://www.fsf.org/philosophy/free-software.html>, last visited: 20/01/01.

³³ For the rationale behind this strategy see: R. Stallman, 'Selling Free Software', Free Software Foundation, <http://www.fsf.org/philosophy/selling.html>, last visited: 20/01/01.

- they may cover costs associated with digital media, duplication or distribution
- they may relate to complementary proprietary software bundled with the open source code package
- training and support services may be charged for
- users may choose to pay a fee to ensure they receive a particular brand of open code software from a reputable distributor.

Whether open code software can be successfully charged for is largely a function of its utility and complexity. Generally the higher or complexity the more likely that payment will be possible. However, once open code software becomes widely distributed (which, due to the fact that software is digital media, is generally not very long) payment tends to become a moot point - users will generally be able to acquire free copies of the software from those who have previously purchased them. This situation is greatly facilitated by the Internet.

*Copylefted software: "copyleft - all rights reversed"*³⁴

Within the broader group of open code software is a subgroup, copylefted software. Most open code software is 'copylefted' although a small portion is distributed under different terms. Generally, when people talk of 'open code software' they are actually referring to copylefted software. Unless otherwise specifically noted, this note adopts the same convention in all subsequent discussion.

Copylefted software is open code software whose intellectual property model does not permit individuals who redistribute or modify the software to add any additional restrictions to the distribution terms they received the software under when they. Put differently:

copyleft is the rule that when redistributing a program, you cannot add restrictions to deny other people the central freedoms [see above]. This rule does not conflict with the central freedoms; rather it protects them.³⁵

The effect of the copyleft intellectual property model is to ensure that anyone who redistributes copylefted software, with or without modification, must pass to subsequent users the same freedoms they themselves enjoy. Hence, copyleft preserves access to source code plus the central freedoms for all users, regardless of how far along the distribution chain they are from the original developer. It ensures that all users receive the software with the freedom to run and copy and modify it without restriction.

The rationale behind 'copylefting' software is simple: to prevent individuals along the distribution chain from modifying copylefted software, copyrighting it in their own name, and then distributing it under their own terms and restrictions; the effect of which would be to proprietise what until then had been 'free' software. Copyleft effectively prevents "free rider" behaviour and ensures rapid and unrestrained distribution and modification of software without fear that it will be subsequently commercialised. It is, at base, a protection mechanism.

Copyleft is the invention of Richard Stallman of the Free Software Foundation. Stallman believed that copyright had become a tool of oppression in the hands of software developers and that the best way to defeat this practice was to use the law against itself. He argues that:

³⁴ R.Stallman per T. Stanco, 'Looking at the General Public License and Open-Source Licenses, Linux Programming, http://www.linuxprogramming..._story.php3?tsn=2000-07-10-001-03-ID-UU, last visited: 05/12/00.

³⁵ Supra note 32.

the licenses of most software [developers] are designed to take away your freedom to share and change [software]. By contrast, [copyleft] is intended to guarantee your freedom to share and change software.³⁶

Copyleft uses copyright law, but flips it over to serve the opposite of its usual purpose: instead of a means of privatising software, it becomes a means of keeping software free.³⁷

Copyleft is in fact a specialised form of copyright that employs specialised license terms to ensure that the central freedoms and access to source code is guaranteed for all subsequent distributions.

Normal copyright asserts ownership and identification of the author, ... prevents intentional distortion of the work by others and prevents destruction of the work. But it also carries other restrictions - such as restricting the reproduction or modification of a work.

Copyleft contains the normal copyright statement, asserting ownership and identification of the author. However, it then *gives away* some of the other rights implicit in normal copyright: it says that not only are you free to redistribute this work, but you are also free to change the work. However, you cannot claim to have written the original work, nor can you claim that these changes were created by someone else. Finally, all derivative works must also be placed under these terms.³⁸

To copyleft software the legal owner first copyrights the software, then adds copyleft distribution terms in the form of a licensing agreement, which stipulates the terms under which the software may be run, modified and distributed. Copyleft creates a legal instrument that effectively makes the copylefted software legally inseparable from the central freedoms.

Subsection 3: The Gnu Public License

The freedoms associated with open code software are preserved by the copyleft distribution terms rather than the software's underlying copyright. Like its commercial cousin, copyleft distribution terms are found in the license that accompanies the software. The bulk of copylefted software is distributed under the Gnu Public License or GPL.³⁹ The GPL is a further invention of Richard Stallman. It is in effect the codification of the principles of copyleft software; it contains the necessary distribution terms and formalities to create a legally effective license that protects open code software and enshrines copyleft freedoms. The GPL has been extraordinarily successful in achieving Stallman's goal of encouraging the distribution and protection of open code software. It is the most widely used of all open code licences and most major open code projects utilise it, including Linux. Open code software distributed under the GPL is represented on the software taxonomy diagram as a subgroup of copylefted software.

The Open Source Definition

To understand open code software it is necessary to examine the terms of the GPL. Reproduced below are the principal terms of the Open Source Definition or OSD - the accepted layman's summary of the terms of the GPL. The OSD is used as

³⁶ Supra note 34.

³⁷ Richard Stallman, 'The GNU Project', Free Software Foundation, <http://www.fsf.org/gnu/thegnuproject.html>, last visited: 05/12/00.

³⁸ M. Stutz, 'Applying Copyleft To Non-Software Information', Free Software Foundation, October 2000, http://www.FreeSoftwareFoundation.org/philosophy/nonsoftware_copyleft.html; last visited: 22/01/01.

³⁹ For a full text of the Gnu Public License see Appendix 1 of this note.

a definition and a set of criteria to evaluate whether or not a particular piece of software may be called open code software. Software that fails to meet OSD criteria is not recognised as open code software by the open code community.⁴⁰

1. Free Redistribution

*The license may not restrict any party from selling or giving away the software [either by itself or] as a component of an aggregate software distribution containing programs from several different sources. The license may not require a royalty or other fee for such sale.*⁴¹

Comment: The GPL empowers the user to make any number of copies of the software and protects the user's freedom to choose to redistribute the software either gratis or for a fee. The user is not required to seek the permission of any other individual or organisation to redistribute the software. This means that the software can be sold but, regardless of whether the seller modified the software, purchasers are free to redistribute it as they wish.

The requirement for free redistribution prevents short term profit taking in favour of long term development. Regardless of the modifications a developer makes to an open code project, they can derive revenue only from sales made personally by them to first generation users. Once sold, the software can be freely redistributed amongst users thus substantially reducing the sale value (but not the utility) of all subsequent distributions. The free redistribution term effectively acts to prevent 'defection' of open code developers to proprietary based software models.

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-publicised means of downloading the source code, without charge, via the Internet. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a pre-processor or translator are not allowed.*⁴²

Comment: Access to the source code is the central freedom of the open code intellectual property model generally. Access to source code is an essential requirement if developers are to study or modify the software. Source code must be in its most accessible form and preferably distributed with the software itself to make modification and development as easy as possible.

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.*⁴³

Comment: This is the 'engine provision' of the GPL - it has two key effects:

(1) it requires that the licensor of the work to grant all subsequent licensees the right to make modifications of any sort and distribute derivative works. This is essential if the software is to be developed and evolved by subsequent developers.

⁴⁰ For a general discussion of open source licenses see: R. Gomulkiewicz, 'How Copyleft uses License Rights to Succeed in the Open Source Software Revolution and the Implications for Article 2B', *Houston Law Review*, 36, 179, [1999].

⁴¹ The Open Source Definition, v 1.7, Open Source Initiative, <http://www.opensource.org/osd.html>; last visited: 22/01/01.

⁴² Ibid.

⁴³ Ibid.

(2) All derivative works must be distributed under the terms of the original licence - preventing the addition of restrictive terms that would defeat the purpose of the license. This second clause is the genius of the copyleft intellectual property model because it enables continual public evolution of open code software stock.

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.*⁴⁴

Comment: The GPL requires that the author of a particular piece of source code be acknowledged. Commonly this is achieved by retaining the author's copyright notice on the original code as it is passed along the distribution and development chain. The rationale for this term is two fold:

(1) users of the software have a right to know who is responsible for the code they are using.

(2) a key motivation to develop open code software is the reputation derived from one's peers within the hacker community.⁴⁵ For credit giving to take place the code author's name or sigil must be specifically attributable to the relevant code. Without this reputation-based incentive, little open code software would be developed.⁴⁶

However, in certain circumstances authors of code may not wish to be associated with derivative works. For example, a hacker may wish not to have her name associated with what she considers to be substandard code that is subsequently appended to her original code. The GPL avoids this situation, and continues to enable modification by others, by requiring (at the discretion of the developer) that the source code distributed with the software be pristine base source code (the original or root source code) and that all subsequent code modifications accompany the software as 'patches.'⁴⁷ In this way it is easy for users and developers alike to track the history of changes to a particular project, to pick and choose among different versions, and receive credit (or criticism) for each individual's addition to the code pool. In addition, the GPL enables the developer of a particular 'flavour' of software to preserve a particular name or trademark in relation to that flavour.

5. No Discrimination Against Persons or Groups

*The license must not discriminate against any person or group of persons.*⁴⁸

Comment: To provide the maximum benefit possible from / to open code software the maximum number and diversity of persons and developers should be encouraged to use the software. Accordingly no discrimination may be made against users

⁴⁴ Ibid.

⁴⁵ For an excellent discussion of the economics and social-interactions of hacker communities see: E. Raymond, 'Homesteading the Noosphere', <http://www.tuxedo.org/~esr/writings/homesteading/homesteading-6.html>; last visited: 28/12/00.

⁴⁶ Ibid.

⁴⁷ Patch: An addition to a piece of code, usually as a remedy to an existing bug or misfeature. A patch may or may not eventually be incorporated permanently into the program. (definition); <http://foldoc.doc.ic.ac.uk/foldoc/foldoc.cgi?query=patch>; last visited: 04/02/01. Using patch files does not add to the burden of distributing or assembling source code. Free programs are available that automatically merge patches into the main source, and these programs can be instructed to run automatically when extracting a source package. Linux distributions such as Debian and Red Hat use this procedure for all of the modifications they make to the software they distribute.

⁴⁸ Supra note 41.

of open code software, no matter how laudable the justifications for the discrimination. Further, because of term 3 (above) of the licence any restrictions would be self-replicating within license, raising the possibility that justifications for discrimination may lapse while the restrictions continue long afterwards. For example: in the 1980's software distributed under license from the University of California, Berkeley, prohibited use by the South African police force. While apartheid has fallen the licence continues in effect.⁴⁹

6. No Discrimination Against Fields of Endeavour

*The license must not restrict anyone from making use of the program in a specific field of endeavour. For example, it may not restrict the program from being used in a business, or from being used for genetic research.*⁵⁰

Comment: On a similar rationale to term 5, no restrictions may be placed on the use of the software in a particular field of endeavour - the software must be equally usable in an abortion clinic or by an anti-abortion organisation.

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.*⁵¹

Comment: The GPL is intended to take affect automatically without the necessity of a signature. The term also prevents restricting GPL covered software by indirect means, for example by requiring the signing of a non-disclosure agreement prior to use.

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.*⁵²

Comment: Term 8 prevents indirect restrictions on the use of open code software by prohibiting terms that would make that software usable only if used in conjunction with another product. The software must remain free if you separate it from the software distribution it came with.

9. License Must Not Contaminate 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 code software.*⁵³

Comment: This term is concerned with aggregation (including two or more programs on the same media) not derivation (incorporating two or more pieces of code into the same program). It prevents overly dogmatic approaches to other forms of software that would hamper the spread of open code software.

⁴⁹ B Perens, 'The Open Source Definition', Open Sources: Voices from the Open Source Revolution, O'Reilly Publishing, 1999.

⁵⁰ Supra note 41.

⁵¹ Ibid.

⁵² Ibid.

No Warranties

Although not an official term of the GPL, most open code software is distributed under license terms that provide the software “as is” and specifically “disclaim both warranties of product and performance and that the software is non-infringing of the copyright, patent or trademark rights of others.”⁵⁴ The purpose of warranty provisions is to shift the legal risk away from open code developers, who are generally individual volunteers who cannot afford the insurance premiums borne by large development corporations.

Taken together, the terms of the Open Source Definition, as expressed in the Gnu Public License, provide a legal mechanism that protects and encourages the evolution of open code software. The driving force behind this license is to ensure that users have access to source code so that they can continually test and improve software and distribute those improvements to the community. Stepping back from the detail of the GPL, it is clear that the aim of the open code licensing model generally is to benefit user-developers, by ensuring code cannot be propertised. The open code model is the other pole of the dialectic discussed earlier between commercial and open code software. The two stand in sharp contrast to one another across a middle ground that is rapidly gravitating to either pole. The dialectic between them is developing and growing ever more taut as open code grows in popularity. This tension between developer control and user freedom is discussed in greater detail in Part II, Section B, Subsection 1.

The purpose of Part I of this paper was to clearly define open source code and its underlying licensing model. Now at the end of Part I we can look back and identify the two key features of open code software: freely available source code (the technical distinction) and the GNU Public License (its unique intellectual property model). With this definition now in place, several more questions pose themselves: why is open code software growing in popularity, succeeding even, despite massive resistance from traditional software developers? Secondly, a Generation X question: why should we care? What does it matter which model or other comes to dominate the software industry? The answers to these questions will be the subject of Part II.

⁵³ Ibid.

⁵⁴ Supra note 34.

PART II - WHY WE SHOULD CARE ABOUT OPEN CODE SOFTWARE.

Why should we care about open code software?

There are many possible answers to this question but at base they can be reduced to the following statement: *because it is important*. But this begs the question, why is it important? Understanding this ‘why’ is the difficult part. The answer is twofold:

- (1) it is important because of its *development model*. The open code development model generates a number of key efficiencies that make open code software superior in many respects to commercial alternatives.
- (2) it is important because of its *implications*. Open code software’s emphasis on user freedom and empowerment, combined with its unique development model has implications that spread deep into the legal and political landscape of our society, a society increasingly dependant on thinking machines.

These two factors are far from independent. The legal and political implications of open code software occur largely as a function of its development model. Hence if we are to understand these implications we must first consider the development model that drives them. Accordingly, Part II adopts a bipartite structure:

- Section A will analyse why the open code development model is important, as well discuss its history and examine the development process itself, keeping in mind its downstream implications.
- Section B will examine two of open code software’s key implications: firstly its potential effect on intellectual property law, secondly, the possible impact on modern governance regimes.

This structure will enable us to carefully and comprehensively answer the overarching question we have set for ourselves: why is open code software important?

SECTION A: THE OPEN CODE DEVELOPMENT MODEL

Why is the Open Code Development Model Important?

In Microsoft’s Halloween Documents, author Vinod Valloppillil assessed the threat represented by the open code software development model or *process*:

OSS [Open Source Software] has a unique *development process* with unique strengths. The OSS process is....
[an] interesting, non-replicable asset which should be thoroughly understood...

Commercial software development processes are hallmarked by organisation around economic goals. However, since money is often not the (primary) motivation behind Open Source Software, understanding the nature of the threat posed requires a deep understanding of the process and motivation of Open Source development teams.

In other words, to understand how to compete against OSS, we must target a *process* rather than a company.⁵⁵
[emphasis added]

Valloppillil recognised in open code software what few before him in the commercial software industry in 1998 had yet come to grips with: open code software was not *really* about the code itself *per se* (though the ‘openness’ of the source code, technically and legally, is undeniably important). What was critical is that behind open code software stood a revolutionary new *development model*: a unique *process* that presented a substantial threat to the established order. Eric Raymond, in critiquing Valloppillil’s observation made the following comment:

This is a very important insight, one I wish Microsoft had missed. The real battle isn’t [Windows] NT versus Linux - it’s closed source development versus open source. The cathedral versus the bazaar.⁵⁶

But why is this development model important to us, approaching as we are from a legal perspective?

- First, because it is the final piece of the puzzle that, together with open source code and the GPL, explain the true nature of open code software.
- Second, because the open code development model generates efficiencies that have caused many to regard open code software as superior to commercial alternatives. This ‘superiority’ has attracted a rapidly growing user base, a user base that drives open code software’s legal and political implications. Put simply, if no one used open code software it would have no downstream implications.
- Third, because the open code development model has a direct impact on the legal ownership of open code software. It prescribes a form of software ownership that has significant legal and political implications, implications that are at the heart of the second and third issues central to this paper.

We begin by looking at where the open code development model came from and its current situation before turning to an analysis of the development process itself and its key efficiencies.

Subsection 1: The Long March of the Open Code Movement

Though some in the New Economy would have us believe otherwise, open code software is not especially ‘new’. It has existed since software coding first began in the 1960’s and until the early 1980’s it was the natural state of the then still infant computer industry. During this period, computer programmers (known as ‘hackers’)⁵⁷ freely shared source code amongst one another. The notion of ‘open’ or ‘closed’ source code was yet to develop:

We did not call our software "free software" [open code software], because that term did not yet exist; but that is what it was. Whenever people from another university or a company wanted to port and use a program, we gladly let them. If you saw someone using an unfamiliar and interesting program, you could always ask to see the source code, so that you could read it, change it, or cannibalise parts of it to make a new program.⁵⁸

⁵⁵ Supra note 23.

⁵⁶ This is a reference to Raymond’s highly influential work “The Cathedral and Bazaar” per E. Raymond, ‘Critique of the Halloween Documents’, 1998, <http://www.opensource.org/halloween/halloween1.html>; last visited: 15/01/01.

⁵⁷ For those who still have misconceptions regarding the term ‘hacker’ please see: <http://www.tuxedo.org/~esr/faqs/hacker-howto.html>, last visited: 11/01/01.

⁵⁸ Supra note 37.

The commercialisation of the computer industry in the late 1970's saw a radical shift in this environment. A new software paradigm emerged, closed code software, in which closed teams of developers were harnessed to create proprietary software that was then sold for profit. The story of the young Bill Gates exemplifies the period: Gates was nearly expelled from Harvard for using publicly funded labs to develop commercial software. When required to make his code publicly available, he left Harvard and founded Microsoft. Gates' success blazed a trail that other developers followed, resulting in the general dissolution of the collaborative software communities that had previously existed.

Despite the collapse of the open code community, Richard Stallman, a former MIT Artificial Intelligence Lab hacker, refused to accede to the newly dominant proprietary paradigm. In 1983 Stallman founded the Free Software Foundation to encourage the development and distribution of high quality, open code software. 'Free' in this sense referred to the user's freedom to read and or modify the source code that accompanied the software, not the price of the software, which might or might not be sold for profit. In Stallman's words: "'think free speech', not 'free beer'".⁵⁹ Stallman's objective in founding the Free Software Foundation was to:

bring back the co-operative spirit that prevailed in the computing community in earlier days - to make co-operation possible again by removing the obstacles... imposed by owners of proprietary software.⁶⁰

Stallman developed a series of software programs collectively called the GNU Project.⁶¹ In its early years the GNU project enjoyed little commercial recognition and was virtually unknown outside hacker circles. However, during this period Stallman did achieve one key success: the development of the GPL.⁶² The GPL would later become crucial by enabling the widespread distribution of open code software in such a way that prevented it from being hijacked by commercial interests and propertised for profit.

Despite the genius of copyleft, 'free software' remained at the fringes of the computer world until 1992, when the GNU project was integrated with the kernel⁶³ of a Unix based operating system⁶⁴ written by University of Helsinki student Linus Torvalds. The result was an extremely robust, commercial grade operating system that could be redistributed for free (bundled with its source code) and hence could be read and modified at the user's will. This new system was called Linux.⁶⁵

From 1992 onwards Linux began to acquire a dedicated following of users and developers. In 1998 Eric Raymond⁶⁶ founded the Open Source Initiative to meet what he felt was a failure by the Free Software Foundation to effectively promote open code software. This change in strategy fortuitously coincided with Linux's following reaching a critical mass. In 1998 Linux exploded into the public arena, as mainstream commercial interests suddenly realised the extraordinary value

⁵⁹ Supra note 32.

⁶⁰ Richard Stallman per David Bollier, 'The Power of Openness - A Critique and a Proposal for the H2O Project', <http://eon.law.harvard.edu/opencode/h2o>, last visited: 22.12.00.

⁶¹ Stallman named the software GNU as a recursive acronym for "GNU's not Unix" - Unix being the dominant operating system of the time.

⁶² GNU Public Licence - the details of GPL are discussed in detail above in Part I, Section B, Subsection 3.

⁶³ A kernel is the essential part of a Unix or other operating system, responsible for resource allocation, low-level hardware interfaces and security; (definition) <http://foldoc.doc.ic.ac.uk/foldoc/foldoc.cgi?kernel>, last visited: 12/01/01.

⁶⁴ The low-level software which handles the interface to peripheral hardware, schedules tasks, allocates storage and presents a default interface to the user when no application program is running. An operating system will be split into a kernel which is always present and various system programs which use facilities provided by the kernel to perform higher level house-keeping tasks, often acting as servers in a client-server relationship. Some operating systems include a graphical user interface and window system, others do not. The facilities an operating system provides and its general design philosophy exert an extremely strong influence on programming style and on the technical cultures that grow up around the machines on which it runs; http://foldoc.doc.ic.ac.uk/foldoc/foldoc.cgi?operating_system, last visited: 12/01/01.

⁶⁵ Linus + Unix = Linux.

in this extremely stable and secure operating system, that could be freely redistributed and modified. Twenty years after it had been abandoned, open code software had resurfaced.

At the time of this writing it is estimated that there are between 4 and 27 million Linux users world wide,⁶⁷ with most estimates in the 10 - 12 million range.⁶⁸ Linux is growing at a phenomenal rate: 1999 figures indicate business usage increased 212% over the year, while general usage is doubling roughly every 12 months - a rate four times higher than average.⁶⁹ Linux is now the second most popular server operating system⁷⁰ after Windows NT with a 25% market share.⁷¹ Its growth rate in the server market outstrips the growth rates of all other operating systems combined.⁷² Its popularity as a desktop operating system⁷³ is still only 4%,⁷⁴ trailing behind Microsoft's dominant market share of 87%, however, this is barely a year after the first GUI⁷⁵ interface for Linux was released.

Linux's success has had two major effects. Firstly, it has proved to mainstream industry players that the technical quality of open code software is equal or superior to existing commercial alternatives. Barely three months after Linux burst into the open, a confidential Microsoft internal memorandum (the infamous Halloween Documents)⁷⁶ made explicit the real threat Linux posed:

a key barrier to entry for OSS [open source software] in many customer environments has been its perceived lack of quality. Recent case studies provide very dramatic evidence in customer's eyes that commercial quality can be achieved / exceeded by OSS projects... Linux represents a best-of-breed UNIX, that is trusted in mission critical applications, and - due to its open source code - has a long term credibility which exceeds many other competitive [operating systems.]⁷⁷

In addition, Linux drew attention to what until then had been a little appreciated fact outside hacker circles: the critical role open code software plays in the infrastructure of the Internet. The 'running gear' of the Internet, the basic code and software that powers the Internet, is built almost entirely from open source code. Fundamental Internet code such as the Domain Name System,⁷⁸ sendmail,⁷⁹ TCP/IP stacks⁸⁰, Usenet,⁸¹ and electronic mailing lists⁸² are all open code. Further, much of the

⁶⁶ See Eric Raymond's Homepage, <http://www.tuxedo.org/~esr/>; last visited: 04/02/01.

⁶⁷ FAQ About Open Source, Open Source.org, <http://www.opensource.org/faq/html>, last visited: 23/12/00.

⁶⁸ Supra note 37.

⁶⁹ J. Alami, 'WebTrends to Offer High-End Web Analysis and Reporting to Red Hat Linux Users', Linux.com, <http://www.linux.com/newsitem.phtml?sid=1&aid=4457>; last visited: 12/01/01

⁷⁰ Server Operating System: an operating system that runs on a server - a computer which provides some service for other computers connected to it via a network.

⁷¹ S Shankland, 'Linux sales surge past competitors', CNET.com, <http://news.cnet.com/news/0-1003-200-1546430.html?tag=st.ne.1002.bgif?st.ne.fd.gif.j>, last visited: 12/01/01.

⁷² Supra note 69.

⁷³ A desktop or 'client' operating system is designed for consumer use on typical desktop and laptop computers.

⁷⁴ By way of comparison, Apple's MacOS operating system enjoys a 5% market share.

⁷⁵ GUI (Graphical User Interface): software that allows the user to operate a computer using clickable icons rather than text - for example Windows 95 or Apple's MacOS system.

⁷⁶ Leaked to Eric Raymond of the OpenSource.org, Microsoft has publicly acknowledged the authenticity of the memorandum.

⁷⁷ Supra note 23.

⁷⁸ The Domain Name System maps numeric IP addresses "199.201.243.200" to domain-style (human readable) addresses "www.netaction.org" enabling users to navigate using names and words instead of 12 digit numbers. The DNS system is almost totally dependent on a free software package known as BIND (Berkeley Internet Name Daemon). <http://www.netaction.org/articles/freesoft.html>, last visited: 19/01/01.

⁷⁹ Approximately 80% of all email handling and delivery on the Internet is accomplished using Eric Allman's open source code sendmail program, <http://www.netaction.org/articles/freesoft.html>, last visited: 19.01.01.

active content of the Internet (fill-out forms, animations etc) is written in the open code programming language PERL, that considerably outperforms commercial competitors.⁸³ Arguably a greater success even than Linux, but less well known, is Apache Web Server software.⁸⁴ Apache powers over 60% of all web servers, well ahead of its nearest rival Microsoft, whose IIS server software holds only 19%.⁸⁵ The ability of these programs to reliably scale with the Internet's massive growth is a key indicator of open code quality: as one internet engineer put it: "the 'ah-ha' for open source software came to me when I realised, 'wait, open source *is* the Internet.'" ⁸⁶

Secondly, once the corporate world realised that open code software was highly reliable and could compete with commercial alternatives the commercial potential became obvious. Key players such as IBM, Oracle, Corel, Informix and Sun announced they would port applications to run on Linux systems, while a number of start-ups including RedHat, Suse, and Caldera forged new business models based on open code software distribution and support.⁸⁷ Organisations such as Boeing, Northern Telecom and NASA as well as universities in France, Australia, Eastern Europe and China adopted Linux as their preferred operating system; citing reliability, savings in licence fees⁸⁸ and versatility as key reasons to shift from proprietary alternatives.⁸⁹ Then the unthinkable began to occur: commercial software developers began to release the source code of major programs, of which the most famous example to date is the Netscape Browser. Since that time open code software has continued to attract more users and developers, significantly outpacing the growth rates of the commercial development model.

In sum, what Linux has achieved is a perceptual reorientation within the IT world. Government and business have slowly come to realise that the commercial development model of the last two decades no longer holds a monopoly on the software industry, that there is a realistic and credible (non-commercial) alternative. It was able to achieve this reorientation due to a truly remarkable development model that we turn to now.

Subsection 2: The Open Code Development Model

Linux turned traditional closed software development methodology on its head. The traditional model mandated small developer teams (frequently ten or fewer in number) working closely together to a pre-prescribed development plan. Developers would develop code, then laboriously trawl through it searching for bugs, before releasing beta versions of the software every 4 - 8 months. A small group of beta testers would then run the software for several weeks searching for overlooked bugs and functionality issues. Responses would be collated, studied and debated before the core development team made the necessary changes to the code base. The software would then be re-tested by developers, and re-released for

⁸⁰ A TCP/IP stack contains the network protocols that enable communication between different types of computers and computer networks. It is one of the fundamental building block of the Internet.

<http://www.computeruser.com/resources/dictionary/TCP/IP>, last visited: 19/01/01.

⁸¹ The collection and distribution of news articles on the USENET is managed by INN, an open code software package running on thousands of sites worldwide, <http://www.netaction.org/articles/freesoft.html>, last visited: 19/01/01.

⁸² The open code software Majordomo Mailing List package is the most popular list manager on the Internet, <http://www.netaction.org/articles/freesoft.html>, last visited: 19/01/01.

⁸³ Sun Microsystems' first Webmaster, Hassan Schroeder, called PERL "the duct tape of the Internet.", <http://www.netaction.org/articles/freesoft.html>, last visited: 19/01/01.

⁸⁴ When you click on a link in your desktop browser, you are actually sending a request to web server software running on a remote web site asking it to send you content to display. Over 60% of these web servers run Apache server software.

⁸⁵ Netcraft Web Server Survey, <http://www.netcraft.com/survey/>, last visited: 19/01/01.

⁸⁶ L. Lessig, 'Open Code and Open Societies: Values of Internet Governance' Harvard Berkman Centre for Internet and Society, <http://cyber.law.harvard.edu/works/lessig/opensocd1.pdf>, last visited: 19/01/01.

⁸⁷ Supra note 60.

⁸⁸ For example, savings in licence fees over hundreds of computers motivated the producers of the movie 'Titanic' to adopt Linux-based computing to produce the special effects for the film. 'The Software Gold Rush', Cware, http://www.cwareco.com/gold_rush.html; last visited: 02/01/01.

⁸⁹ Supra note 60.

another round of debugging. This release and revision cycle might continue for months until the software was judged ready for public distribution. The unchallenged assumption behind this methodology was Brooks' Law.⁹⁰ Fred Brooks', a developer at IBM in the 1960's, had theorised that adding additional developers to a software project increases the complexity of the project at a quadratic rate, resulting in decreasing productivity. Since its formulation in the 1960's Brook's Law had occupied a position of gospel truth in the development world, effectively constraining software development to small, highly ordered teams of professional developers working in splendid isolation. Eric Raymond, in his analysis of the open code development model in 'The Cathedral and the Bazaar',⁹¹ characterised the traditional development methodology as 'cathedral building', because of its emphasis on centrally controlled teams working to a structured and pre-defined plan.

Conversely, Raymond likened the open code development model to a bazaar, a chaotic meeting of hundreds of minds, discussing, arguing and debating; all working independently but in concert toward a loosely defined but realisable goal. This new model emphasised factors that flew in the face of established development wisdom:

- massive development teams [in excess of a thousand contributing developers]
- no pre-defined development plan beyond a general, accepted goal
- extremely loose organisational structure with no official management hierarchy
- work contributed entirely on a voluntary basis

By all rights any project utilising such a model should have collapsed into a screaming and indiscernible heap - especially a project as complex as the development of an operating system. But Linux did not, and it has come to represent the paradigmatic example of the open code development model. In effect, Linux *is* the open code development model. Therefore it is to Linux that we now turn.

The founder of Linux, Linus Torvalds, began with a simple but ambitious goal: to create a functional and reliable operating system that could be freely used by other hackers. Torvalds started with a non-proprietary Unix-based kernel called Minix,⁹² which he began to modify to achieve his goal. The key difference between this and any other hobbyist hacker project was that Torvalds frequently and deliberately posted the results of his developments on the Internet and communicated them to fellow hackers and friends. The embryonic Linux operating system quickly attracted a small following and it was not long before people were offering to help develop the code or unilaterally submitting patches to bugs they had independently encountered.

It was at this point that Torvalds' true genius emerged and that the open code development model was born. The genius lay in Torvalds' management of the Linux project rather than any piece of code. Torvalds responded to the growing support enthusiastically by taking six key initiatives:

1. Utilised the ability of the Internet to create a virtual development community

The Internet's ability to create virtual communities in cyberspace is well known today, but one of the first groups to appreciate this unique feature were hackers. In 1991 Torvalds realised that the Internet enabled the creation of 'virtual

⁹⁰ "'Adding manpower to a late software project makes it later' -- a result of the fact that the expected advantage from splitting development work among N programmers is $O(N)$ (that is, proportional to N), but the complexity and communications cost associated with co-ordinating and then merging their work is $O(N^2)$ (that is, proportional to the square of N)." F. Brooks, "The Mythical Man-Month", Addison-Wesley, 1975 per <http://www.user.xpoint.at/michael-otte/jargon/html/entry/Brooks's-Law.html>; last visited 04/02/01.

⁹¹ Supra note 28.

⁹² Ibid.

development communities': hundreds of hackers communicating with one another instantly, swapping ideas, code and information regardless of geography. Whereas in the 1970's hacker communities had been restricted to the individuals at a single research lab, now the global hegemony of english, and the infrastructure of the Internet, enabled hundreds of developers from different countries, institutions, and development backgrounds to pool their resources quickly and effectively. Torvalds used the Internet to achieve the unthinkable: create a highly effective software developer community measured in the thousands.

2. delegated as much responsibility for code development as possible, and then some.

Delegation runs completely contrary to the traditional development model because it inevitably sees the project head off the safe confines of the development plan and into uncharted territory. But this is precisely what Torvalds wanted to encourage. Freed from restrictive development plans, Linux's volunteer developers followed whichever particular whim, interest, or function that attracted them. The outcome was a massive increase in productivity and innovation as developers worked on what they wanted to, rather than what they were paid to. The result was self-selected specialisation, the most economically efficient form of production possible.⁹³

3. massively increased the pace of releases and the incorporation of user feedback.

While quick release cycles were not unknown to the hacker community, Torvalds massively scaled the intensity of the release cycle up to match the complexity of the project - the more complex the project, the more releases. At the height of development in 1991 Torvalds was releasing a new kernel every 24 hours - a pace unimaginable to traditional models where release cycles were measured in months not hours.⁹⁴ The result was massively accelerated development as hackers contributed code and then literally watched the operating system come together before their eyes.

4. actively cultivated the Linux developers base, treating them as co-developers rather than 'testers'.

Torvalds treated the huge pool of Linux hackers as equals, as 'co-developers'. This was in total opposition to the traditional disdain reserved for 'testers' under the closed code development model. These co-developers repaid Torvalds thousands of times over by ardently pawing through each new release version, looking for bugs, reporting them back to the Linux community and frequently suggesting fixes and or submitting patches themselves. By keeping the developer base continually stimulated Torvalds was able to bring a massive resource of developer man-hours to bear against the problem of debugging the evolving kernel. Eric Raymond dubbed this strategy 'Linus' Law':

Given a large enough beta-tester and co-developer base, almost every problem will be characterised quickly and the fix obvious to someone. Or less formally "given enough eyeballs, all bugs are shallow."⁹⁵

Linus' Law goes to the very heart of the difference between the closed and open code development models. Under the closed code model debugging is a long and complicated process that absorbs months of developer time, resulting in long release cycles which frequently fail to produce a bug-free product. By contrast, the open code model treats de-bugging simply as part of the continuing evolution of the code, code which is constantly screened and tested by thousands of developers trying to find the elusive bug that will get their name into the development logs. The result is extremely stable and almost perfectly bug-free software.

⁹³ D. Kaminsky, 'Core Competencies: Why Open Source is the Optimum Economic Paradigm for Software', Doxpara Research, [wysiwyg://20/http:doxpara.netpedia.net/core.html](http://www.doxpara.net/pedia.net/core.html); last visited: 22/12/00.

⁹⁴ Supra note 28.

⁹⁵ Ibid.

5. was “open to the point of promiscuity.”⁹⁶

Torvalds encouraged everyone and anyone to become involved in the Linux project, regardless of technical background. His motivation was partly philosophical, reflecting the 1970’s developer ethic that software development should be an inclusive rather than exclusive process, and partly focussed on the need to expand the developer base as quickly as possible. This policy:

- substantially increased the growth of the developer base
- increased the level of creativity and innovation
- cultivated a decidedly democratic atmosphere which helped to prevent in-fighting
- helped to rapidly spread the ‘Linux word’ and promoted Linux evangelisation even among non-contributors

6. made every attempt to give credit where credit was due.

Most hackers hack for the sheer enjoyment of it and the recognition they receive from their peers within the hacker community. Creating a particularly innovative piece of code or finding and fixing an especially difficult bug earns respect from community members. However, there is a catch to this practice: in order to earn respect, the developer must publicise their work, and the only way to do so is to *give it away*. In this respect the hacker community fits the model of what anthropologists describe as a ‘gift community’ - a community in which respect is earned not by what one personally accumulates, but by what one publicly contributes to the community. Put simply, hackers hack code and then give it away to increase their reputation.⁹⁷ Torvalds recognised that to secure the co-operation of the hacker community every effort had to be made to ensure that those who contributed code were publicly identified and acknowledged.

Subsection 3: The Result - Self Specialised Distributed Resource Processing

To the surprise of many Linux did not collapse under the weight of inefficiencies massively multiplied across hundreds of developers. Rather, the operating system that emerged in 1992 demonstrated none of the undiscovered bugs and security flaws that had dogged the releases of other operating systems.⁹⁸ Instead Torvalds and his motley (if large) band of developers had managed, entirely with geographically distributed labour to produce an operating system that was more than a match for commercial alternatives. Linux, with an estimated R&D cost at over US\$2 billion contributed entirely from volunteer labour worked *extremely* well.⁹⁹

What Torvalds had created was a revolutionary new development methodology: a highly efficient self specialised distributed resource processing system, in which geographically distributed developers work independently, but in parallel, via decentralised decision-making processes toward multiple development goals, taking any number of distinct routes to get there. The most effective routes are incorporated into the overall code base and the process repeated, resulting in the rapid evolution of the code base simultaneously across multiple fronts of development. The process is extremely efficient and capable of bringing massive development resources successfully to bare against even the most complex problems.¹⁰⁰ In fact

⁹⁶ Ibid.

⁹⁷ For a detailed discussion of the hacker community and gift cultures see E. Raymond, ‘Homesteading the Noosphere’, *supra* note 45.

⁹⁸ See for example the problems encountered by users of Windows 95 and NT. There are approximately 250MG of official bug patches for Windows NT in distribution.

⁹⁹ *Supra* note 93.

¹⁰⁰ Linux proved this by successfully tackling the most complex project it could have: the development of an operating system from the ground up.

a number of theorists have argued that in many significant respects the open code model closely reflects the processes of evolutionary biology and is capable of generating similar efficiencies, except over a vastly shorter time frame.¹⁰¹

The outstanding success of the Linux project created a template that was emulated by subsequent open source projects such as Eric Raymond's 'Fetchmail'.¹⁰² The Linux development model became *the* open source development model, inspiring hackers to go open source, to marshal distributed programming resources to generate substantial efficiencies in software design and development. As this paper is a legal work it is not its intention to examine open code software's efficiencies in detail - unfortunately much that could be said here must go unmentioned. Rather, we will briefly discuss the three principal efficiencies while the remaining efficiencies will be listed for the sake of brevity.

Key Efficiencies:

- **massively stable** Linux has consistently outperformed virtually every other rival operating system in terms of reliability and stability. Linux systems crash or stall less often and have been known to run for years, 24 hours a day, without failure.¹⁰³
- **highly scalable** Linux performs as effectively with a small number of clients as a large number. This is particularly important in the context of the Internet where server loads may fluctuate significantly over a short time period.
- **extremely secure** Linux systems are significantly more difficult than proprietary systems to crack.¹⁰⁴ When security holes are found the open source development model tends to fix them much faster - in 1997 when a major cross-system security hole was discovered, Linux distributors had fix patches distributed within hours, closed source operating systems didn't react for months.¹⁰⁵

Related Efficiencies:

- **developer efficiencies** - increased development speed
- lower development costs
- lower legal costs
- risk spreading (across multiple developers)
- cost sharing (across multiple developers)
- **user efficiencies:** - increased reliability

¹⁰¹ "Decentralized decision-making processes, in the language of complexity theory, are powerful algorithms for finding "high points on the fitness landscape" (i.e., solutions to problems defined over complex, interdependent spaces). The problem-solving power of decentralised systems is well-documented and reasonably non-controversial in mathematical, physical, and biological systems, underlying phenomena as diverse as parallel processing algorithms in computational mathematics and natural selection in the design of living things." D. Post, 'Of Black Holes and Decentralised Law Making in Cyberspace', <http://www.temple.edu/lawschool.dpost/blackhole.html>, last visited: 25/03/00. For more detail on the relationship between evolutionary biology and software development see: D. Post, 'Chaos Prevailing on Every Continent' Chicago-Kent Law Review, Symposium on the Internet and Legal Theory, v73, n4, 1998, pp.1055 - 1101, see also: A. Oliva, 'The Competitive Advantage of Free Software', 1st Free Software International Forum 2000, May 2000, <http://www.ic.unicamp.br/oliva.html>; last visited: 28/12/00.

¹⁰² Supra note 28. For an extensive list of open code software projects see: the GNU Project, <http://www.fsf.org/home.html>; last visited: 04/02/01.

¹⁰³ 'The Business Case for Open Source', OpenSource.org, <http://www.opensource.org/for-suit.html>; last visited: 23/12/00.

¹⁰⁴ Cracking a system involves breaching its security protections to gain access to protected data or operations. 'Crackers' are individuals who use their knowledge of programming to break into computer systems, as distinguished from 'hackers' who use their knowledge productively to develop or modify systems.

¹⁰⁵ Supra note 51.

- increased scalability
- increased security
- decreased legal compliance costs¹⁰⁶
- effectively infinite product life time¹⁰⁷

These efficiencies are mentioned here, albeit briefly, because they help to explain why businesses, governments and users have adopted open code software. This is important because as the open code user base grows and as it captures more developer mind share, it begins to generate profound implications that ripple out through the software industry and into society generally. Implications for intellectual property and implications for governance - the two issues that are at the heart of this paper. It is to these implications that we now turn.

SECTION B: THE IMPLICATIONS OF OPEN CODE SOFTWARE

The implications of open code software are many: technical, legal, political, economic, social and cultural. Unfortunately exploring the breadth and depth of these many implications is beyond the short scope of this paper. Instead our focus will be restricted to two of these six areas of influence: the legal implications of open code software and the implications for democratic governance. These two areas are respectively the subject of Subsections (1) and (2) of this second half of Part II. Within each subsection we will examine a single significant issue from among the many different potential implications:

- Subsection 1 considers the nature of intellectual property law and the key role that open code software may play in restoring balance to a regime that has been the subject of ill considered reform in the last two decades.
- Subsection 2 draws on our knowledge of the open code development process to analyse how this may affect the ownership regime of open code software and the possible implications this regime may have for democratic governance.

Subsection 1: Intellectual Property

Intellectual Property - A Background

To understand the implications of open code software for intellectual property in the 21st century requires that we have an understanding of intellectual property generally. Intellectual property is of three basic types: copyright, patent and trademark. In keeping with the theme of this note to date we will focus our attention on copyright, although many of the conclusions reached are equally applicable to patent and trademark.

Common Law 'intellectual property' begins in the 1400's, shortly after Gutenberg's invention of the printing press. The English Crown, concerned to control dissemination of information and generate additional revenues, issued 'Letters Patent' to favoured organisations granting a monopoly over the printing of certain works. For almost three hundred years this intellectual property regime would be a source of financial abuse and censorship power for the English Crown.

¹⁰⁶ With open code software users need only be familiar with a single well known license, with stated objectives and no hidden motives - the GPL. This decreases the legal costs associated with purchasing new software.

¹⁰⁷ With closed source software if a developer discontinues a product then users of that product are eventually forced to change to alternative products because maintenance patches, updates and hardware syncing is no longer possible. However, with open code software because the source code is available the user themselves can take responsibility for software maintenance.

By 1780 the Statute of Anne was finally ending the corruption of the intellectual property regime in England. However, on the other side of the Atlantic, the framers of the United States' Constitution were wrestling with its poisoned legacy. Basking in the afterglow of the Enlightenment, they were extremely attuned to the critical role that thoughts and ideas would play in the infant republic's future. However, the corruption and censorship of the English regime was still fresh in many minds. Intellectual property would become one of America's fiercest constitutional debates, pitting two of its greatest minds, Hamilton and Jefferson, against each other.

The details of this debate are beyond the scope of this note, what is critical is the outcome. The framers reached a compromise: property in intellectual creations would be enforced by the state, but only where such enforcement produced a demonstrable and substantial benefit to the common good. This balance was enshrined in the Constitution - article I, section 8, clause 8 provides that Congress shall have the 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.¹⁰⁸

This clause is the basis of the United States' Copyright and Patent Acts and has had an enormous influence on the intellectual property regimes of much of the common law world. The clause is unusual in that it is the only grant of power to Congress in the Constitution that has a stated purpose. This underscores the explicit role that the founding fathers saw for intellectual property: copyright and patent were *limited* monopolies that were tolerated *only* as a means to an end: to provide an incentive to authors to undertake intellectual endeavour.

The immediate effect of our copyright law is to secure a fair return for an author's creative labour. But the ultimate aim is, by this incentive, to stimulate artistic creativity for the general public good.¹⁰⁹

In this we can clearly see the utilitarian impulse that dominates American intellectual property philosophy,¹¹⁰ an impulse that has been seriously abused in recent years.¹¹¹ This aside, the key feature of the clause is the concept of *balance*: the framers made every attempt to ensure that there was a balance between the monopoly rights granted and the creative endeavour produced. This balance is succinctly summarised by Judge Walker:

copyright law seeks to establish a delicate equilibrium. On the one hand, it affords protection to authors as an incentive to create, and, on the other it must appropriately limit the extent of that protection so as to avoid monopolistic stagnation.¹¹²

The First Congress was careful to follow the framers' balanced view. The Copyright Act gave authors the exclusive right to publish and sell maps, charts and books. Other types of writings were not regulated. Those that were regulated were not restricted as to use, other than the prohibition on publishing and sale. Hence one could copy, translate or create derivative works from these writings. The initial copyright term was 14 years. At the end of this period the copyright ceased and the copyrighted material returned to the commons, for the enjoyment of all members of the public without restraint.

¹⁰⁸ Constitution of the United States of America, Article I, section 8, clause 8.

¹⁰⁹ L. Loren, 'The Purpose of Copyright', Open Spaces Quarterly, December 22 2000, <http://www.open-spaces.com/article-v2n1-loren.php>; last visited 29/12/00.

¹¹⁰ P. Samuelson, 'Economic and Constitutional Influences on Copyright Law in the United States', http://www.sims.berkeley.edu/~pam/papers/Sweet&Maxwell_1.html; last visited: 29/12/00.

¹¹¹ For more details, see below in Part II, Section B, Subsection 2 under 'The Impact of Intellectual Property Laws'

¹¹² Judge Walker, Second Circuit, per L. Loren see supra note 118.

The situation is vastly different today. The utilitarian impulse has metamorphosed and is now cited as authority for the prevailing, though naively simple, view that intellectual property rights are a simple linear function. This view argues that the more and larger the rights that are granted, the greater the incentive for authors to create and consequently the greater the number of books, works of art, computer programs and movies that will be produced. This view has been heavily fortified by natural rights theory that argues that an author's work should enjoy protection on the grounds that is the fruit of their creative endeavour.¹¹³

The effect of these shifting impulses has been a steadily increasing preoccupation in the United States with protection of the author's intellectual 'property' against 'theft' or 'misuse' by others. Since the 1980's the intellectual property rights of authors have been radically expanded by a series of laws that increasingly threaten the 'delicate equilibrium' of the intellectual property regime. Copyright no longer attaches solely to maps, charts and books but reaches anything "fixed in a tangible medium of expression,"¹¹⁴ including software. It has been enlarged to encompass not only works, but also derivatives of that work, so that they too fall under the original copyright. Further, the copyright term has been massively extended from 14 years, to the life of the author plus 70 years or 95 years for corporations.¹¹⁵ By way of example, this means that the Windows 95 copyright will not expire until 2090. When Windows returns to the commons in 90 years time it is doubtful there will even be a computer primitive enough to run it.

The rise of the Internet has only served to accelerate this process. Terrified by the prospect of a medium that enables virtually limitless copying and distribution of digital media, key American content owners such as the major film studios, Microsoft and Time Warner AOL have desperately sought ways to fix this 'bug' in the architecture of cyberspace. Their efforts have not gone unrewarded. In 1997 Congress passed the NET or "No Electronic Theft" Act¹¹⁶ that made it a *criminal act* for a person to wilfully infringe a copyright by reproducing or distributing the work, even if that person received no profit or commercial advantage.¹¹⁷ A year later the President signed the DMCA (Digital Millennium Copyright Act)¹¹⁸ that provides for both civil and criminal sanctions against those who reverse engineer measures employed by copyright owners to control access to their works. These sanctions apply even where such circumvention is for the purpose of enabling a use that would otherwise be legal i.e. for 'fair uses'.¹¹⁹

This expansion of intellectual property law is especially concerning as it tends to affect products in markets in which network economic effects dominate, i.e. software, communication products and the Internet.¹²⁰ In such circumstances even a relatively insignificant or limited intellectual property right can provide a crucial advantage, that when leveraged by network economic effects, generates a powerful monopoly. Bill Gates was one of the first to appreciate this fact with his

¹¹³ For an excellent discussion of economic, constitutional and natural rights influences on United States copyright law see supra note 110.

¹¹⁴ L. Lessig, 'The Limits of Copyright', *The Standard*, <http://www.thestandard.com/article/display/0,1151,16071,00.html>; last visited: 23/12/00.

¹¹⁵ Sonny Bono Copyright Term Extension Act, 1998; S 505, P.L. 105-298, 11 Stat. 2827.

¹¹⁶ No Electronic Theft Act; 105 P.L. 147; 111 Stat. 2678; 1997 Enacted H.R. 2265; 105 Enacted H.R. 2265

¹¹⁷ This overturns a century old tradition that a profit motive was a requirement for 'criminal infringement'.

¹¹⁸ Digital Millennium Copyright Act, Pub. L. No. 105-304, 112 Stat. 2860 (1998).

¹¹⁹ For an excellent discussion of the impact of the DMCA see: P. Samuelson, 'Intellectual Property And The Digital Economy: Why The Anti-Circumvention Regulations Need To Be Revised', http://www.sims.berkeley.edu/~pam/papers/Samuelson_IP_dig_eco.htm#ftn20; last visited: 31/01/01; and also: A. Lutzker, 'Primer On The Digital Millennium Copyright Act', <http://www.arl.org/info/frn/copy/primer.html>, last visited: 31/01/01.

¹²⁰ See supra note 25.

dream of 'a PC on every desk' - a dream that has become a reality in which Microsoft is the target of anti-trust litigation by the Department of Justice for anti-competitive and monopolistic practices. Hence, recent intellectual property reforms serve only to further amplify the potential leverage that intellectual property rights can generate in what are now economically critical markets.

The principal beneficiaries of these reforms (and coincidentally, those that have been instrumental in shifting the underlying impulses behind intellectual property law - does anyone remember the "PIRACY - IT'S A CRIME" warnings on their last video rental?) are the major content owners. These organisations argue that ideas can be made *property* and that, like any other property owner, they are entitled to protection from those that 'steal' property. They have lobbied government and made the following case: "computers and the Internet have made it easier for people to 'steal' from us - therefore we deserve more protection." In the words of James Boyle:

[corporations] see the Internet as a giant copying machine, a threat to content providers rather than an opportunity for them. Indeed, if there is a theory behind [their] curious vision of copyright law, it seems to be this: more copying equals more copyright violation, thus it is necessary to increase copyright protection as compensation for declining revenues.¹²¹

Such arguments have largely been successful. Governments around the world, not simply the United States, have moved to increase the protection enjoyed by content owning industries. The result has been a radical expansion in the size and scope of intellectual property rights. An expansion in what, reduced to its base, amounts to a state sanctioned monopoly given to private enterprises for the purposes of promoting private profit creation.¹²²

As if these legal protections were not enough, these same corporations have invested billions of dollars in new technologies known as 'Technical Protection Measures' or TPM. TPM are programs that supplement legal protections by providing owners with even greater control over the use of their content. At the owner's discretion they can protect content by preventing it from being copied, distributed or modified, regardless of the users legal right to do so. TPMs make it safe for these owners to widely distribute their code, and profit from that distribution, without fear that it will be 'pirated' by users. TPMs allow copyright holders to:

direct how often a book can be read, or by whom; they can control whether or what parts can be copied, or on what machine the books can be read¹²³

Because the TPM is a part of the actual product itself and is not legally restricting the user's rights, it is not subject to any of the legal mechanisms that ensure the protection of user rights with 'normal' non-TPM works. Instead the creator of a TPM protected work may restrict access to the work as much or as little as the market for that work is prepared to bear. So, in keeping with the example above, if I buy a book I can read it as often as I want. I can copy parts of it out for my own use. I can sell it to a friend. The publisher of the book has no control over my ability to do any of these things and under the Fair Use doctrine I have the legal right to do them. However, if I download an electronic book protected by a TPM then I might only be able to read it once before the TPM erases it from my hard drive. It does not matter that I have a right *in law* to read

¹²¹ J Boyle, 'Intellectual Property Policy Online: a Young Person's Guide', 10 Harvard Journal of Law and Technology 47 (1996) per: <http://www.law.duke.edu/boy/lesite/joltart.htm>; last visited: 31/01/01.

¹²² The author points out with some considerable irony that until 1891 foreign works enjoyed no intellectual property protection in the United States' whatsoever. For the first century of the republic's history it was the most well known and most despised pirate nation on Earth.

¹²³ Supra note 121.

it multiple times or copy parts of it text or sell it to a friend - if I physically cannot do it, it then I cannot do it, legal right notwithstanding. Of course, if I was a skilled hacker then I might be tempted to disengage the TPM in order to enjoy those rights that I am legally entitled to (such as reading the book multiple times) and that I would be enjoy if it was a normal book. But, as mentioned above, the passage of the Digital Millennium Copyright Act specifically prevents such actions and makes it criminal and civil offence to do so.¹²⁴ An act whose principal supporters were the major content holders in the United States.

The Impact of Intellectual Property Reforms

The effect of these initiatives has been a radical shift in the balance of intellectual property law, in favour of content owners and to the detriment of the common user and future creators. At its most basic level this shift has been driven by an underlying assumption that can be paraphrased in the following manner:

We live in a democratic capitalist state. The foundation of our legal and political systems is property. Private citizens can own property and can exchange it in a free market. Ownership and exchange of property generates progress. The more ownership and the greater the exchange, the more progress we achieve. The more perfectly we can protect property, the easier it can be owned and exchanged. The easier it can be owned and exchanged, the more it will be owned and exchanged. Therefore, more perfectly protected property equals more progress.¹²⁵

What two decades of reforms have achieved, unwittingly or not, is to create *more perfectly protected property*. Property that cannot be copied or modified without the owner's consent. Property that can be freely exchanged with anyone, anywhere, without fear that it will be stolen or copied or modified. Property that can be perfectly owned and perfectly exchanged.

Perfect property implies perfect control of that property. Control vested in the owner of the property, *not* the user. Thus what we are really discussing here is control and power. The last twenty years have witnessed a shift in the power dynamic between the user and the owner of intellectual property. We have seen a slow but steady transference from the former to the latter, as owners have accumulated increasing control over intellectual property assets. The effect has been to radically recast the delicate balance established by the Constitution between owners and users. The scales have tipped heavily in favour of intellectual property owners.

Now you might ask: what's the problem? It's their property after all - don't they have a right to protect it? What does it matter if this archaic balance is now an imbalance?

It matters because intellectual property is not solely an output. It is also an input. Modern societies both *produce* and *consume* intellectual property. If a society is to produce creative works it must possess an abundance of creativity that can be freely drawn upon. In short, it must have a lively and vibrant commons. A *commons*. A place, in the words of Lawrence Lessig, that "modern political culture has forgotten".¹²⁶ In our headlong rush to (perfectly) propertise everything intellectual, we have forgotten that the commons is critical to the flourishing of the very thing we most seek: intellectual property. A rich

¹²⁴ Supra note 118.

¹²⁵ This construction owes much to Lawrence Lessig see L. Lessig, 'Open Code and Open Societies', Keynote Address: Free Software - A Model for Society, Tutzing, Germany, June 2000, <http://cyberlaw.stanford.edu/lessig/content/index.html>, last visited: 04/02/01.

¹²⁶ Ibid.

commons is an essential ingredient in a 'knowledge economy', because it enables individuals and organisations to 'stand on the shoulders of giants'¹²⁷ rather than building from the ground up on every new initiative.

The key feature of the commons is not simply that it exists but that:

Anyone can draw from the commons *without the permission of anyone else*. These resources exist in a place where anyone in society is free to draw upon them.¹²⁸ [emphasis added]

Without the permission of anyone else. There can be no control, no restriction placed on the use of commons' resources. They are free to be used by anyone, in any manner so desired, without the permission of anyone else. This freedom is precisely that which encourages and enlightens creativity within society. We need only consider a scenario in which the commons did not exist to realise its profound importance. Anytime you wished to use an idea that was not entirely original and your own you would be required to ask permission, or pay a fee, or meet any number of conditions required by the owner of whichever work the idea was 'based' upon. The stifling impact of such a scenario is all too apparent. Who would become a writer if she had to pay a fee to use one of the five basic plots? Lessig describes this situation as the 'closed society'

The closed society... where the few control access for the many; where the few control content. Where to use, or play, or criticise, or share content you need the permission of someone else. Where the commons has been shrunk to nothing. Where everything to which you have access, you have access because you have asked the permission of someone else.¹²⁹

Yet this nightmare scenario is closer than one might actually think. The past two decades of reform, the enthusiastic 'protection' of intellectual property, has seriously eroded the boundaries of the commons. We have been blinded by a naïve misconception that intellectual property is a linear function, that by extending ever more protection to content owners we will produce more works. In fact, all we have created is perfect property for commercial enterprises. Property that will effectively, if not in fact,¹³⁰ never enter the commons. Property that will never be a source of inspiration for anyone, unless of course you can pay, and if you do have a good idea, well, then you can expect to pay handsomely. By way of example, if Shakespeare were writing today and the rights to his works were assigned to Time Warner AOL then the copyright would not end until sometime in the 22nd century. The adaptations, interpretations, parodies and even educational study that goes on today could, at the discretion of AOL, simply be prohibited or charged for, at whatever fee AOL desired. In essence, as intellectual property becomes ever more perfect we dam the stream that feeds the commons and as we do so we slowly but surely strangle society's creativity. Intellectual property is an input and as any economic text book will tell you, if you raise the cost of an input then you must either produce less or charge more - both ways the user loses.

So why, if these are the consequences, has this been allowed to occur? Because we have forgotten (some would say been distracted from) the fact that the original purpose of intellectual property was solely "to promote the Progress of Science and the useful Arts"¹³¹ rather than protect owners. Instead we have become transfixed on what is an inherently short term and ultimately destructive view of intellectual property:

¹²⁷ Sir Isaac Newton, last visited: 1727

¹²⁸ Supra note 125.

¹²⁹ Ibid.

¹³⁰ It is theoretically possible that by using Technical Protection Measures alone an individual or organisation might prevent the public from ever having free access to the protected content. If the content was not brought under any intellectual property regime then there would be no requirement to bring it into the commons even at the end of copyright term.

¹³¹ Supra note 108.

If [intellectual property] monopolies granted to copyright holders are larger then obviously the content owning industry can generate more revenue. This increase in revenue results in a larger gross national product, a smaller trade deficit and greater employment... however, the threat of monopolistic stagnation looms very real¹³²

It has been allowed to occur because powerful interests in our society that have the capacity to shape public opinion and lobby government have focussed on short-term profit and the survival of an outdated status quo, at the expense of long term damage to society's intellectual environment. The foolhardiness of this policy in an age dominated by innovation and knowledge is startling.

*The Penguin to the Rescue*¹³³

Faced with such a gloomy prospect, the question we now ask is: how does open code software change this situation? The answer is obvious. Above all else, open code software is about freedom and user sovereignty: the freedom to use software, the freedom to copy it, the freedom to distribute it, the freedom to modify it. It exists in the commons: it is free for anyone to use without need of permission from anyone else. In fact, open code software is even *freer* than an idea in the commons because it can never be propertised, nor can any work derived from it - it must always remain free as a constant source of inspiration for any who would use it.¹³⁴ Open code software returns to the user the freedom and sovereignty that have been progressively eroded away over the previous two decades. On an individual level it is helping to re-establish the balance between user and owner. That this is its effect is to be expected - open code software is designed to be 'free software', this was its original name and its stated purpose. Stallman founded the Free Software Foundation to restore to users the freedom the commercial development model had erased. He used the intellectual property model of commercial software as a 'legal negative' to produce what he felt would be 'positive software', software that increased rather than decreased the user's sovereignty. The continued development of GUI systems¹³⁵ for open code software, such as GNOME and KDE for Linux, is only likely to further accelerate this trend as open code software becomes increasing user friendly and penetrates the desktop market more effectively.

Open code software is growing like a benevolent virus,¹³⁶ spreading quicker and further than any other form of software ever before. Why? Not just because it is better and more efficient¹³⁷ but because it is FREE in all senses of the word. Why pay for a system that comes with dozens of restrictions and conditions when you can download for free a system that you can use in anyway you like? This is where the 'superiority' of open code software and its development model discussed above has its impact. For as open code becomes ever more popular, the inexorable gears of market competition slowly begin to turn, forcing commercial software developers to back away from perfect control of software. Instead they are being forced to make, what for many, is an agonising choice:

- resist the new impetus and sit King Canute-like to watch their market share slowly disappear beneath the growing open code tide [a la Microsoft]; or

¹³² Supra note 109.

¹³³ Tux, the Linux mascot, is a penguin.

¹³⁴ This is due to the effect of the GPL see Part I, Section B, Subsection 3.

¹³⁵ Supra note 75.

¹³⁶ Or malevolent depending on your perspective.

¹³⁷ See above, Part II, Section A, Subsection 3.

- begin to offer the same degree of freedom that can be obtained for free in five minutes on the Internet [a la IBM]¹³⁸

The process is unlikely to occur quickly and all in probability will be fought tooth and nail by many in the industry. But it has begun and the unique features of the open code development model, its powerful evangelising potential and its resistance to proprietization, mean that it will be extremely difficult to stop or reverse. Open code software sits as like an enormous weight on other end of the intellectual property scales that had become so tilted. It is forcing, by the weight of accumulated users, software developers to abandon perfect control in favour of a more balanced approach to intellectual property rights in software. It has set in motion forces that will end an era of monopolistic dominance by big development firms, who have dictated the terms and conditions under which users could use software they had legally purchased.

Now you could say “so what, open code software is just that *software* - there are plenty of other mediums in which perfect control is still a possibility.” Now this statement seems true, that is until one looks carefully at the direction our economy and society is developing. When we do, we see two startling impulses:

1. software is everything. Or at least it soon will be. There are few aspects of our economy and society that are yet to be touched by computers and the software that runs them. In the realm of intellectual property this is now virtually a fait accompli: DVDs, MP3s, JPEGs, GIFs; all intellectual property, all forms of traditional media (movies, music, images), all now digitised, all now powered by software. Software that lets you play, copy and edit depending on what the writer intended. Which software? Well if you don't have a choice already you soon will have. One only needs to look at DeCSS¹³⁹ and the background to the Kaplan decision to see the direction things are headed and how open code software will influence that path.¹⁴⁰

2. open code is not the only ‘threat’ to the established order. The open code philosophy of freedom and user empowerment has been adopted by a new generation of hackers who are creating new ways to share intellectual property. The most startling of these new initiatives is peer to peer software, of which the free music swapping program Napster is the pre-eminent example. Although Napster is not itself open code software its effect on intellectual property has been similar. It has rendered existing laws and methods of control irrelevant. The monopolies that the major record companies have held over the reproductions of musicians’ works for the past seven decades have amounted to nothing as music has been freely and massively traded by millions of users world wide. Neither is Napster alone, dozens of other such sites have sprung up,¹⁴¹ many employing software that is significantly more sophisticated and difficult to prohibit than Napster.¹⁴² Some of the most effective of these new programs are open code initiatives, written by dozens of users around the world, impossible to track, impossible to shut down. The cumulative effect of these new initiatives is that any form of digital media desired, even specific files or programs, can now be obtained quickly, efficiently and freely.

¹³⁸ A. Leonard, ‘How Big Blue Fell for Linux’, Salon.com, http://www.salon.com/tech/fsp/2000/09/12/chapter_7_part_one/index.html; last visited: 01/02/01.

¹³⁹ DeCSS is program developed by Norwegian student Jon Johansen that allows the user to unlock the CSS encryption the protects DVD discs and hence enables the DVD files to be copied. This represents a serious threat to the film industry’s attempt to establish DVD as a new standard because it means that DVDs can be pirated in a similar fashion to movies released on video.

¹⁴⁰ Judge D. Harvey, ‘Electronic Civil Disobedience: Universal City Studios v Reimerdes - Liability for Linking and the Provision of Information’, September 2000.

¹⁴¹ For an overview of such sites see: ‘Pooling Resources’, The New York Times, May 10, 2000.

¹⁴² A good example here is the open code software program GNUtella, which unlike Napster, has no centralised operations at all - the entire program is distributed over each and every one of its user’s geographically distributed computers. Consequently, there is literally nothing for authorities to target to stop digital media piracy using GNUtella.

The Digital Age that began in the 1990's promised much to the major content owning industries. They could not possibly resist the potential inherent in content digitisation: the promise was simply too rich and the competitive consequences of clinging to old technology too damning. But as the 1990's advanced, content owners began to realise that had grasped a two-edged sword: just as digital content was so much easier to produce, distribute and sell, so was it so much easier to copy, steal and modify. What the Digital Age has in fact ushered in is a massive challenge to the established intellectual property order. Open code software is likely to be at the forefront of this challenge for three reasons:

1. as discussed above software is everything - digital media is controlled by software and therefore it is software that will become the weapon of choice in the battle to 'control' or 'free' content.
2. it is likely to be *open code* software because the nature of the open code development model (decentralised, geographically distributed, grass roots based, highly evangelised and steeped in philosophies of freedom and libertarianism) creates precisely the conditions that are most likely to generate software-based resistance to establishment sponsored content control. Open code developers are infinitely more likely than commercial developers to undertake initiatives that are illegal or violate intellectual property rights because they are so much harder to catch and have much less to lose.¹⁴³
3. from a technical perspective, because open code software develops extraordinarily faster than other forms of software it tends to find the 'gaps' or 'holes' (legal or illegal) in existing content protection regimes or mechanisms before other forms of software do.

If open code software continues to grow at its present rate it will force a massive and radical rethink of intellectual property law. There are two projected outcomes:

1. *Liberalisation*. Those among government and commerce who realise the full implications of the growing open code movement will not necessarily view it as a threat. Like any new technological advance open code software creates winners and losers, and a particular government or corporation's reaction to the advance frequently determines which group it will fall into. Open code software represents a *forced* swing back towards a more balanced intellectual property regime by *de facto* empowerment of individuals. There have already been initiatives on the part of a number of major commercial players to make this movement *de jure* by easing the level of control exercised over key products and intellectual property assets. The examples of IBM and Netscape are prominent here but even Microsoft, the bogeyman of the open code movement, has made veiled references to such a strategy.¹⁴⁴ The strategy behind these initiatives is to capture a strong position early on, in what increasingly looks to be a potential paradigm shift in the software industry. Although we have yet to see a full flowering of the "if you can't beat 'em, join 'em" strategy, the first movements do suggest that such a strategy may play an increasingly important role in corporate intellectual property development in the future.

The key question of whether governments will be inclined to adopt a 'liberalisation' policy is more difficult to answer. It will depend in part on the reaction and influence of the major content players within their jurisdiction, the nature of their existing intellectual property regime and the success of legislative actions taken to date. In general though, based on observed reactions through the 1990's one is inclined to predict governments are likely to become more illiberal rather than

¹⁴³ For development of this point see Part II, Section B, Subsection 2.

¹⁴⁴ The Halloween Documents made the following suggestions as potential responses to the threat posed by open code software: "Capture parallel debugging benefits via broader code licensing - be more liberal in handing out source code licences to NT to organisations such as universities and certain partners... Put out parts of the source code - try to generate hacker interest in adding value to MS-sponsored code bases". See supra note 23.

less. Liberalisation is most likely to occur in situations where increased intellectual property protection has failed, hence opening a path to less restrictive responses.

2. *The Crackdown*. Exhorted on by declining revenues among major content owners, governments may attempt to 'crack down' on 'pirates' by fortifying existing laws and passing ever more draconian intellectual property protections. This process has already begun in the United States as we have discussed above, with the passage of the NET, DMCA and Copyright Term Extension Acts. A major feature of this response will be attempts at international legal harmonisation of intellectual property regimes as a response to the globally distributed phenomenon of intellectual property piracy. We already witnessing such attempts with the continuing development of the WIPO treaty regime.

Professor Michael Froomkin, creator of the theory of 'regulatory arbitrage'¹⁴⁵ argues extremely persuasively that a great 'Internet irony' is looming, in which the liberalising tendencies of the Internet and digitisation of media, provokes a highly undemocratic back lash from vested interests and governments trying to save their collective economic skins.

[the] great looming Internet irony: what was intended and promoted as the great anarchistic, liberating, democratising technology may in fact spur a reaction so strong as to make the world significantly less democratic. If indeed governments and vested interests conclude that the only way in which they can maintain their position is to band together with like-minded counterparts in other countries to enact multilateral treaties or vest increasing power in supra-national organisations, the power of existing democratic institutions will be reduced world wide.¹⁴⁶

This will ultimately become an Internet governance debate. National governments will 'win' if collectively they are able to develop effective means to 'govern' the Internet and the activities that take place there. But in the opinion of the author, government is largely too late - but not for the reasons one might at first suspect. Although there are undoubtedly very powerful governance regimes that can be put in place,¹⁴⁷ what we are really seeing here is not just a technological revolution but also a social one. We are witnessing the coming of age of a generation of individuals, the 'Napster junkies', who are more than willing, in the words of their parents, to "F**K the draft". The draft forty years on being attempts by governments and powerful corporations to restrict the free flow of information in our society. These are computer literate and extremely net savvy individuals who have to date demonstrated a remarkable degree of indifference to the intellectual property regimes of their parents' generation. Just what the impact of this social change will be is beyond the scope of a note dedicated to the legal analysis of software, but regardless it is a factor deserving of more study.

However beyond the lines of crazed twenty-somethings hammering at the walls of intellectual property regimes with their modems is another more telling, and for us more relevant, reason why governance won't work. It is again open code software, and we turn to it now.

¹⁴⁵ "The multinational nature of the Internet makes it possible for users to engage in regulatory arbitrage - to choose to evade disliked domestic regulations by communicating / transacting under regulatory regimes with different rules. Sometimes this will mean gravitating to jurisdictions with more lenient rules, or perhaps no rules at all; sometimes it will mean choosing more stringent foreign regimes (e.g., those with strong consumer protection laws), when stricter rules are more congenial." M. Froomkin, 'The Internet as a Source of Regulatory Arbitrage', <http://www.law.miami.edu/~froomkin/articles/arbitr.htm> last visited: 04/02/01. See article of same name for further discussion of regulatory arbitrage.

¹⁴⁶ M. Froomkin, 'The Empire Strikes Back', <http://www.law.miami.edu/~froomkin/articles/empire.html>; last visited: 25/03/00.

¹⁴⁷ See generally the works of Lawrence Lessig, <http://cyber.law.harvard.edu/lessig.html>, last visited: 28/01/01 and J. Reidenberg, 'Lex Informatica: The Formulation of Information Policy Rules Through Technology', <http://home.sprynet.com/~reidenberg/reidenberg.html>, last visited: 25/05/00, for the details of such regimes.

Subsection 2: The Implications of Open Code Software for Democratic Governance

To understand how open code software prevents effective governance and, consequently to appreciate its implications for democratic government, we first need to step back a little and acquaint ourselves with the theory of societal governance generally. Put simply, we need to comprehend just how our society is currently governed to appreciate the impact open code software is likely to have on our governance mechanisms. Our aim will be to examine those features of open code that will prevent effective governance and project outwards the potential implications. In this respect Lawrence Lessig's recent work concerning governance theory is highly instructive and will be drawn upon extensively. Lessig asserts that human behaviour is regulated by four 'modalities of constraint': law, social norms, markets, and architecture.¹⁴⁸ Each of these modalities influences behaviour in different ways: law operates via sanction, markets via price, social norms via human interaction, and architecture via the environment.

Behaviour, we might say, is regulated by four kinds of constraints. Law is just one of those constraints. Law... orders people to behave in certain ways; it threatens punishment if they do not obey. The law tells me not to buy certain drugs, not to sell cigarettes without a license, and not to trade across international borders without first filing a customs form. In this way, we say that law regulates. ...

But not only law regulates in this sense. Social norms do as well. Norms control where I can smoke; they affect how I behave with members of the opposite sex; they limit what I may wear; they influence whether I will pay my taxes. Like law, norms regulate by threatening punishment *ex post*. But unlike law, the punishments of norms are not centralised. Norms are enforced (if at all) by a community, not by a government. In this way, norms constrain, and therefore regulate.

Markets, too, regulate. They regulate by price. The price of gasoline limits the amount one drives... the price of subway tickets affects the use of public transportation... Of course the market is able to constrain in this manner only because of other constraints of law and social norms... but given these norms, and given this law, the market presents another set of constraints on individual and collective behaviour.

And finally, there is a fourth feature of real space that regulates behaviour - "architecture." By "architecture" I mean the physical world as we find it, even if "*as we find it*" is simply *how it has already been made*. That a highway divides two neighbourhoods limits the extent to which the neighbourhoods integrate. ... That Paris has large boulevards limits the ability of revolutionaries to protest. These constraints function in a way that shapes behaviour. In this way, they too regulate.¹⁴⁹

Governance of behaviour can be achieved by any one, or any combination, of these four modalities. Law is unique among the modalities in that it can *directly* influence the others. For example, the government can use law to regulate markets by imposing taxes, to influence norms i.e. by funding a public ad campaign against smoking, or affect architecture by enforcing building codes. In this respect law is the most 'self-conscious' of the four modalities in that its prime purpose and effect is regulation, whereas the other three modalities regulate less directly. In turn, law can be *indirectly* influenced by the other modalities. For example, there is no sense to a law that decrees that "all dogs must float three feet above the ground" because the architecture of real space, in this case gravity, prevents it and thus indirectly constrains which laws can be passed. Typically governments use a mix of the four modalities to constrain behaviour within society. Which particular modality is the most effective, and therefore most dominant, depends on the context of the regulation.

¹⁴⁸ L Lessig, 'The Law of the Horse: What Cyber Law Might Teach', <http://cyberlaw.stanford.edu/lessig/content/index.html>; last visited: 04/02/01

In a small and closely knit community, norms might be the optimal mode of regulation; as that community becomes less closely knit, law or the market might become better substitutes. In tenth century Europe, mucking about with architectural constraints might have been a bit hard, but in the era of the modern office building, architecture becomes a feasible and quite effective regulatory technique (think about transparent cubicles as a way to police behaviour).¹⁵⁰

Lessig argues that regardless of the situation in the real world, in cyberspace architecture is the dominant and most effective modality of constraint. But what is the architecture of cyberspace? According to Lessig it is 'code' - software code - that creates the environment of the Internet. Code determines what can and cannot be done in cyberspace. For example, if you wish to access a restricted web site you might be required to enter a password to prove you are who you claim to be. This restriction, this *constraint*, on your behaviour is a function of the software code that runs the web site. If this code was written differently there might be no requirement for a password at all. It is the *code* that determines what you and can and cannot do in cyberspace. Except that in cyberspace anything can be allowed to happen. Unlike real space, in cyberspace there is no predefined environment that restricts the architecture we can build, there is no law of gravity that prevents us from building floating buildings or falling upwards. In cyberspace the environment is neutral: we can build whatever we want to by coding whatever we want to. That is not to say that the other modalities of constraint do not work in cyberspace, but that architecture, in the form of code, is the dominant modality. Thus in cyberspace, as Lessig's famous dictum goes, 'code is law'.¹⁵¹

Now in real space, law doesn't just appear from nowhere - it is written by lawmakers. Our representatives in Parliament or Congress sit on our behalf and make laws. In cyberspace the situation is different: code is written by code writers - hence in cyberspace the lawmakers are the *code writers*. Therefore it is hackers and software developers that determine the law in cyberspace. Thus, if I have a web site in the United States and I don't want people in New Zealand to view it, then I can simply write a piece of code to instruct my web server to refuse connections from IP addresses beginning with '203'.¹⁵² Hence in cyberspace, those who write the code decide what is and is not possible there. The impact of this conclusion should be relatively clear: in cyberspace those who control the code, write the 'laws' that determine what can happen there.

Now this is where the story begins to become extremely interesting. If we think back for a minute we will remember that one of the four constraints, law (in the real space sense), had a unique ability to directly influence the other three modalities, including architecture. For example, if the government wanted to reduce gun fatalities, they might regulate gun makers to force them to only manufacture rubber bullets (regulating the architecture of the gun), thereby reducing the number of fatalities. If the government is able to regulate gun makers, it can just as easily regulate code writers. And if it can regulate *code writers*, then it can regulate *code*, and if it can regulate code, then it can regulate the *Internet* and *anything else incorporating code*. Your electronic blender for example, could be regulated (via regulation of the manufacturer who writes its software) to only run on 'slow'. Your car could be regulated to go no faster than 100 km per hour, or more realistically, your Internet browser could be regulated to prevent the display of pornographic sites.

What we are discussing here is indirect regulation of the Internet or any other form of software by legal regulation of its

¹⁴⁹ Ibid.

¹⁵⁰ Ibid.

¹⁵¹ L. Lessig, 'Code is Law', <http://cyberlaw.stanford.edu/lessig/content/index.html>; last visited: 04/02/01

¹⁵² '203' is the three digit prefix of the 12 digit IP number that identifies internet connections emanating from New Zealand. By blocking any IP number beginning with '203' no one using an Internet connection from New Zealand would be able to access the web site.

code writers. Though law can have little *direct* control over cyberspace or software, it can exert significant *indirect* control by regulating code writers and leveraging the absolute power they enjoy over their creations. Hence, it is the government, ultimately, who can, if they wish, determine what happens in cyberspace or what a particular piece of software can or cannot do. The regulation may be indirect, and at times the route to the desired effect (whatever that is) may be a little torturous and far removed, but governments are not the toothless mutts that the cyber-pioneers once thought they were.¹⁵³ Or are they are?

Whether government can regulate code (and hence, the Internet and software generally) depends largely on one simple question: who owns the code? If the code owner is a private corporation, then the government's ability to regulate is assured. Why? Because the code is the property of someone and if the property has an owner then the government can simply regulate the property via that owner. In the case of software, the owners are usually large commercial developers, developers who have much to lose if they fail to acquiesce to government demands. Therefore they are *easy to regulate*. But, what if the code is not owned anyone, how can the government regulate it? This seems to be a foolish question, until that is, one begins to think about open code software and its development model. The thing to remember from our discussion of the open code development model is that it is *distributed resource processing* - hundreds, if not thousands, of developers individually contributing to a collective and leaderless code base containing millions of lines of ever evolving code. What is the implication here?

Simply put, *no one owns open code software*. Why? The reason is self-evident: the open code development model prevents it. The basis of the model is hundreds of developers individually and voluntarily contributing code to a project that has no single owner.¹⁵⁴ The process is pure decentralised manufacturing: there are hundreds of developers, not one or two, and there is no single umbrella organisation or individual who can claim rights to the code and therefore be considered 'the owner'. Although individual developers may be able to point to specific lines of code within a particular piece of software no one individual ever comes close to writing even a substantial minority of the code of an open code software project - it is truly a work of many.¹⁵⁵ To further compound the issue, source code rarely if ever stays constant in an open code project. It is constantly revised and reconsidered, modified and changed, edited in and edited out. The Minix precursor of the Linux kernel is a prime example: Torvalds received Minix in toto, but by the time the first Linux distribution occurred the Minix code was unrecognisable. Nothing of the original code was left, it had been worked over by hundreds of minds for countless hours improving, changing, editing and adapting. There was nothing left of Minix to attach rights to.

The result - ownerless code - is so implacable, so insistent, it almost seems as though the process was designed exactly was this end in mind: hundreds of developers contributing to a code pool that is constantly changing, constantly evolving, never static for even a single hour. It is a lawyer's nightmare. There is nothing to pin property rights to, no single individual or organisation to point at and definitively state "owner!" The closest one can come to identifying an 'owner' are the copyright holders of the original code base. In the case of Linux this is Linus Torvalds, who holds the copyright to the original Linux kernel. But the copyright only applies to Torvalds' original work, *not* to the work of any subsequent contributors. The nature of the contributions subsequently amalgamated into the Linux code base are such that there is no way that they could fall under the rubric of Torvalds' general copyright. The contributions are almost inevitably too original, too different, to fall

¹⁵³ Prior to 1996 the prevailing view amongst most Internet theorists was that the Internet and most other forms of digital media could not be effectively controlled or regulated by terrestrial governments. This view has been largely discredited by governance theories first propounded by Lawrence Lessig, Pamela Samuleson and Joel Reidenberg.

¹⁵⁴ The code writers are free from any legal obligation or relationship to supply code - they do so entirely of their own free will, so there is no legal basis (for example, employment contract) on which a central organisation or individual could legally lay claim to the code.

¹⁵⁵ It is worthwhile noting that a full Linux distribution contains over 15 million lines of code; see supra note 23.

under the concept of derivative work.¹⁵⁶ Even in the rare circumstances where such additions might be considered derivative works, they are more than likely to fall under the protection of the Fair Use doctrine and therefore be effectively outside the sway of the original copyright.¹⁵⁷ The result is that by the time an open code project nears completion, the original code has been so worked over, so modified, adapted and appended to by so many hands, that the original copyright is virtually meaningless. It is certainly no where near definitive enough to enable the holder to make a claim to ownership over the entire code base.

Further, the legal effect of copyleft and the GPL in particular is to effectively enshrine in law the ‘ownerless’ state of open code software.¹⁵⁸ This is because the GPL takes the original copyright holder’s rights and reverses them, converting them to *copyleft*, which effectively (though not legally) neutralises the holder’s copyright, while simultaneously granting virtually limitless rights of use, modification and distribution to subsequent users. The result is that while the original copyright holder retains the legal copyright to the *original* source code placed under the GPL, all *modifications* by subsequent contributors remain the property of those contributors, until they are inevitably modified also. The result is a legally fractured but technically robust product in which ‘ownership’ is diffused to the point of meaningless among hundreds or even thousands of contributors.

In theory, a particular contributor could point to a series of code lines in a broader open code project and claim “those are mine”. At first glance, this claim raises the spectre of the tragedy of the anti-commons,¹⁵⁹ and the threat that any open code project could be doomed to endless stagnation by the suffocating requirement of seeking the consent of a contributor before taking any substantive action in relation to their contribution. However, if one looks closely at the GPL it becomes clear that Stallman’s license deftly avoids this problem, despite the fact that the license was created 15 years before it was first articulated. The answer lies in the GPL’s prohibition on individuals adding any additional restrictions to the terms under which GPL’d software is distributed. This means that if a contributor wishes to see their code contribution incorporated into, and distributed as part, of the broader project, then the terms of the GPL, and *only* the GPL, must prevail. Thus, the modified code must be distributed under the same terms on which it was received i.e. freely and without additional restriction. Consequently, the contributor, while technically retaining the copyright to the contributed code, loses any *effective* control over it, as it now falls under the terms of the GPL, and hence can be happily used *without restriction* by whosoever wishes to do so. Hence, the tragedy of the anti-commons is avoided.

Of course, a contributor could choose to retain full control over the code by not submitting it to become part of the broader project, but then the contributed code loses any value it might have, except in so far as it is useful to the contributor personally. In practice, most developers who create a useful patch for an open code project almost inevitably contribute it to the project for one of three reasons:

¹⁵⁶ The nature of code writing and software design is that appended or even modified code is frequently so different that it cannot effectively be considered part of the original source code, hence the hacker expression ‘patches’ to refer to pieces of code (however large) subsequently added to a program.

¹⁵⁷ This in fact is precisely the reason that commercial developers use licensing agreements - to prevent the creation of superior derivative works over which they have no effective control.

¹⁵⁸ For the details of the GPL see Part I, Section B, Subsection 3 or Appendix 1.

¹⁵⁹ The ‘tragedy of the anti-commons’ is a theory advanced by Michael Heller which describes a similarly negative but opposite effect to the tragedy of the commons as described by Garrett Hardin in his famous work ‘The Tragedy of the Commons’ see: G. Harrett, ‘The Tragedy of the Commons’, *Science* 162, 1243 (1968). The "tragedy of the anti-commons" occurs in situations when multiple owners each have a right to exclude others from a scarce resource and no one has an effective privilege of use; see M. Heller, ‘The Tragedy of the Anti-Commons’, *Harvard Law Rev.* 111, 621 (1998).

- the dominant economic paradigm in the hacker community is a *gift culture*,¹⁶⁰ by contributing useful patches to an open code project the contributor earns that which she values most, respect.
- A developer gains nothing by withholding a contribution from a project and it costs her nothing to contribute it. But, if she does contribute it, then it is likely to encourage further development in those areas the patch relates to - which are likely to be of interest to the developer.
- the simple benefits of reciprocal gift giving have a strong influence. Submitting a contribution encourages reciprocal gift giving to the broader project on the part of other developers, resulting in a lively and creative intellectual commons that may solve future problems the hacker faces.¹⁶¹

Hence, although each contributor may be able to point to small ‘pieces of the pie’ with her name / copyright on it, they are generally only one of hundreds with equally insignificant rights. Whatever identifiable legal title she might hold provides virtually no control and is certainly not sufficient to ground a general claim to ‘ownership’ of derivative works, well aside from the fact that the GPL prohibits this outright. The situation of the one or two major contributors to each project (such as Torvalds with Linux or Raymond with Fetchmail) is little different. The major contribution made confers power initially, but once the code base is brought under the GPL this power largely disappears and legally the contributor becomes first among equally impotent equals. Their only slightly more substantial rights are similarly incapable of generating general rights of ownership over the project code base. In practice, issues of the legal rights associated with copyright of a particular piece of code are rapidly rendered irrelevant anyway, due to the nature of the open code process itself. The rapid, frequent, and radical reinvention of the code base quickly sees the rights to any particular piece of code dissolve in a swirling pool of millions of lines of continually evolving and changing code.

In sum, the legal impact of the open code development model is really rather simple: *open code software has no owner*. There is no one single individual or organisation to attach property rights to - it is at once owned by hundreds of people and no one at all. Its development model effectively precludes ownership by a single individual or organisation. This is the second of the two key impacts discussed above in Part I, Section B, Subsection 1 that the open code development model has on the legal and governance implications of open code software. If open code software has no owner, if there is no one individual or corporation to regulate, *then it cannot be regulated*. Put simply, *closed code software can be regulated, open code software cannot*.

This conclusion runs completely counter-intuitive to most modern political theory. For 500 years since Hobbes, it has been accepted that property is a key antidote to state power. It is a fundamental tenet of the capitalist system that ownership of private property not only increases the general wealth of society but helps to ensure freedom for its citizens.¹⁶² The feeble private property regime in the USSR has long been pointed to as a key factor in the rise and dominance of totalitarianism in that country.¹⁶³ Yet here we have undeniably, the exception that contradicts the rule. Exactly why, at a deeper level, *intellectual property* produces this Sphinx is a question well beyond the scope of this note, but the impact remains clear to see.

There is a great looming irony here and it is this: open code software is ownerless, in the eyes of an AOL Time Warner

¹⁶⁰ For more details on gift cultures see Part II, Section A, Subsection 2.

¹⁶¹ Supra note 45.

¹⁶² But it should be noted, property does not guarantee that freedom - witness the many despots of the 20th century resident in capitalist economies.

¹⁶³ “France, Russia, China: A Structural Analysis of Social Revolutions” per J. Goldstone, *Revolutions: Theoretical, Comparative and Historical Studies* HBJ, 1986.

corporate executive it is *imperfect* property: it cannot be controlled, you cannot stop someone from copying it or modifying or 'stealing' it. And yet this imperfect property is more resistant to government regulation than the most private property imaginable. In fact the more private, the more perfectly protected by law intellectual property becomes, the easier it is to regulate. Every attempt to create more perfectly protected property is only pushing content owners further away from their customers, the users, and closer to their traditional enemy, the government.

This great irony and the conclusions on which it is based, all depend on the fact the open code software has no owner. Let us imagine for a moment that the government some how figured out a way to identify a single owner of any open code software project, so that the finger could be conclusively pointed and the regulations brought down to bear. The project is regulated and the owner forced to insert some government-mandated code into the program. What would be the effect? Is this the Achilles heel of open code software? Not really, because there is one final trick in the box. This is *open code* software - the source code is available for all to see. If you don't like what you see, well, just change it, because you're free to and so long as you have access to the source code, then no one can stop you. So well the government may be busily forcing the owner to distributed doctored code, the users are just as likely to be distributing patches to remove it. In case more incredulous readers believe this is not the case, the author refers them to the example of the French government and Netscape.

In 1998 Netscape released the source code to the Netscape Communicator Browser, effectively making the program open code software.¹⁶⁴ Netscape established a special site where users could download the source code of the browser and modify it as they wished. A year later the French government approached Netscape Corporation and asked them to modify the browser so that the government would be able to intercept and crack messages sent using its embedded encryption protocol, SSL.¹⁶⁵ Much to the chagrin of the French, Netscape declined the request, not because it would not accede, but because it *could not*. Netscape explained to the French government¹⁶⁶ that even were it to create an indigenous French browser and release it, there was still no guarantee that they would be able to intercept messages, because it depended entirely on which version (the normal or French version) users chose to adopt. Further, even if from that moment forward they distributed only the French version, users would still be able to modify the source code. Even if in desperation Netscape ceased releasing code at all, there were still millions of copies of the source code on the Internet that could be freely modified to remove any unwanted government code. The French were reportedly not amused.¹⁶⁷

The example of the French attempts only reinforces our general conclusion that open code software cannot be effectively regulated by government. Its unique development model renders it ownerless and this, when combined with the fact its source code is, by definition, openly available and can be freely manipulated by users, virtually ensures that it is de facto impossible to regulate with any degree of success. The implications of this finding are enormous. We live in a society in which computers and the software that drives them are becoming essential to every facet of our daily life. The global economy is increasingly predicated on technology and knowledge - the gold and lead of the 21st century. Billions of dollars, billions of lives depend on these things. Underpinning them all is code.

A very select group of organisations control this code. Perhaps two dozen multinational corporations exercise significant

¹⁶⁴ Though not technically because it was still owned by Netscape and was not distributed under the GPL.

¹⁶⁵ SSL, the Secure Sockets Layer, is a security protocol developed by Netscape which "provides data encryption, server authentication, message integrity, and optional client authentication for a TCP/IP connection." Netscape Communications Corp., *Secure Sockets Layer*, <http://home.netscape.com/security/techbriefs-ssl.html>, last visited: 20/12/00.

¹⁶⁶ A process which apparently took some significant time and patience.

¹⁶⁷ L. Lessig, 'The Limits in Open Code: Regulatory Standards and the Future of the Internet', <http://cyberlaw.stanford.edu/lessig/content/index.html>; last visited: 04/02/01.

power over one of the most important resources on this planet. That there are so few, presents an extraordinary opportunity to government to regulate society. Huge and extremely visible, these organisations are highly vulnerable to regulation by government, due to their ponderous size and the immensity of their interests. Faced with a choice between government sanction and the inclusion of mandated code in source code that few users are even aware exists, the choice becomes a fait accompli. What we see here is an ill-starred conjunction: increasingly monopolistic ownership of intellectual property resources, coupled with shrewd governance, enabling control equally as powerful as any direct regulation. More powerful in fact, because such regulation is unseen - it is regulation by stealth. Regulation in ways we do not notice and are unlikely to object to or issue writs against: subtle but undeniable. Indirect regulation by code represents a real and substantial threat to the principals of democracy and open government.

This is not some nightmarish scenario but in fact a very real possibility - the example of the French government above or the recent passage of the Regulation of Investigatory Powers Act¹⁶⁸ in the United Kingdom suggests this is already becoming a reality. A handful of corporations dominate the software development industry and the vast majority of all code written is still closed commercial code. Consider that approximately 90% of all the world's computers run operating systems developed and owned¹⁶⁹ by a single company.¹⁷⁰ Consider that 96% of all Internet browsers, the 'window' that defines how users see the Internet (and increasingly the world), are owned by two companies.¹⁷¹ As the Internet and information technology grows ever more important, indirect regulation by software has the potential to reach out and influence every aspect of our lives: how fast our blender runs, what speed we drive our car, what we can see on the Internet. What we see here in software, are the whispers of hydraulic despotism - a situation that occurs when an individual or organisation wields a near or total monopoly over a critical resource. Such a monopoly has the potential to generate massive social, political and legal power that it is disproportionate to the already substantial economic power wielded.

This potential for unseen, omnipresent regulation on a massive scale occurs because software is closed and because it is owned by just a few. But ironically the very forces that have raised this spectre have created that which can defeat it. Open code software, though still in its infancy, is a powerful antidote to the threat of indirect regulation by software. Because its source code is open, because there are no owners to make demands of, it is impervious to such regulation. As open code software grows more popular it will slowly but surely dilute the ability of government to regulate indirectly. This in itself is a powerful reason to adopt it, aside from any other argument regarding intellectual property or technical superiority. The more that we support open code software, the more that we adopt it and encourage others to do the same, the more we empower ourselves. In a world increasingly dependant on thinking machines, our ability to ensure that they are thinking for us, rather than someone else, is paramount.

¹⁶⁸ Regulation of Investigatory Powers Act 2000 - see <http://www.homeoffice.gov.uk/ripa/ripact.htm>; last visited: 04/02/02 or for critique see: <http://www.privacy.org/pi/countries/uk/>; last visited: 04/02/02.

¹⁶⁹ 'Owned' because it is licensed to you - the user doesn't actually own the software herself.

¹⁷⁰ That company is Microsoft.

¹⁷¹ Netscape and Microsoft - although obviously the fact that Netscape has released the source code to their browser alleviates the situation considerably, despite the fact that Microsoft's Internet Explorer still has over 65% market share.

Conclusion

Open code software promises to play a critical role in the future development of the software industry. It is the only software model in the past two decades to successfully challenge the dominant commercial software paradigm. It is the product of a new development model that has questioned existing development theory and has the potential to revolutionise the software industry. But, as we have seen, the implications of open code software spread well beyond this industry and out into the realms of law and government.

The aim of this paper has been to reveal something of the nature of this unique phenomenon and its impact. As discussed initially, although much writing has been dedicated to technical analysis of open code software, little thought has yet been given to its legal framework or its deeper implications for law and government. Accordingly, this paper chose to break this ground by examining three key legal issues surrounding open code software and its development model. These issues were: firstly, the licensing agreements under which the software is distributed, secondly its implications for intellectual property law and finally, the broader implications open code software demonstrates for democratic governance. In analysing these issues a number of conclusions were reached.

Firstly, by creating a software taxonomy we were able to clearly establish the significant differences that exist between open code software and the dominant commercial software model. These differences stem from two key areas: firstly the fact the source code of open code software is 'open', in that it is freely available to users. Secondly, open code software is governed by a unique and highly innovative licensing model known as copyleft, of which the foremost example in the Gnu Public Licence. This form of licensing guarantees users of open code software almost total freedom to use it as they desire, while faithfully ensuring that such freedoms cannot be stripped from users via commercial propretization. It has been a key factor in the overall success of open code software. This licensing model is a truly unique legal initiative and is worthy of further study in its own right.

Secondly, open code software has drawn attention to the intellectual property reforms of the last two decades. Asses sing the theoretical foundations of intellectual property law it became clear that the law establishes a fine balance between the need to incent the author to create and society's need to have access to a rich and vibrant intellectual commons. In the past two decades intellectual property reforms in the United States have stripped users of key freedoms they have previously enjoyed and endangered our continued ability to innovate and create. Open code software represents a potential antidote to correct this imbalance in intellectual property law, by restoring to users the freedoms they have lost. However, by the same token it is possible that open code software and the Internet may in fact provoke an anti-democratic backlash among governments desperate to protect the intellectual property of the major content owning industries. The implications here are many and varied and, similar to open code licensing, are worthy of more study.

Finally, analysing the open code development model revealed that it has unique legal implications for the ownership regime of open code software. The decentralised and distributed nature of the development process, combined with the effect of the GPL, means that open code software has no effective owner. Consequently, it is extremely difficult to regulate open code software. This stands in sharp contrast to commercial software, which has been rendered easy to regulate due to the intellectual property reforms of the last two decades. It is theorised that the immunity to regulation demonstrated by open code software may play a key role in diluting the ability of government to indirectly regulate society by stealth - a capacity which is represents a serious threat to democratic society.

In conclusion, if we are to say anything about open code software it is that it represents freedom on many diverse levels. It represents the freedom of the user to know what they are using, freedom from burdensome commercial restrictions, freedom from hidden government regulation. In this freedom we find the full flowering of open code software. This paper touched only briefly on these freedoms and those that were not examined are deserving of further study and analysis. In conclusion it is worthy to note that Richard Stallman created free software because he wanted people to be free. In some small way his creation has achieved its promise.

BIBLIOGRAPHY

- Alami J, 'WebTrends to Offer High-End Web Analysis and Reporting to Red Hat Linux Users', Linux.com, <http://www.linux.com/newsitem.phtml?sid=1&aid=4457>; last visited: 12/01/01
- Barlow J, 'Selling Wine Without Bottles - The Economy of the Mind on the Global Net', http://www.eff.org/pub/Intellectual_property/idea_economy.article, last visited: 29/12/00.
- 'Binary file' (definition) Webopedia: Online Computer Dictionary, http://webopedia.internet.com/TERM/b/binary_file.html; last visited: 11/01/01.
- Bollier D, 'The Power of Openness - A Critique and a Proposal for the H2O Project', <http://eon.law.harvard.edu/opencode/h2o>, last visited: 22.12.00.
- Boyle J, 'Foucault in Cyberspace: Surveillance, Sovereignty and Hard-Wired Censors', <http://www.wcl.american.edu/pub/faculty/boyle/foucault.htm>, last visited 15/7/00.
- Boyle J, 'Intellectual Property Policy Online: a Young Person's Guide', 10 Harvard Journal of Law and Technology 47 (1996) per: <http://www.law.duke.edu/boy/lesite/joltart.htm>; last visited: 31/01/01.
- Brooks F, "The Mythical Man-Month", Addison-Wesley, 1975
- 'Categories of Free and Non-Free Software', Free Software Foundation, <http://www.fsf.org/gnu/categories.html>; last visited: 04/12/00.
- Constitution of the United States of America, Article I, section 8, clause 8.
- Digital Millennium Copyright Act, Pub. L. No. 105-304, 112 Stat. 2860 (1998).
- 'Domain Name System' (definition) <http://www.netaction.org/articles/freesoft.html>, last visited: 19/01/01.
- Dyson E, 'Intellectual Value', http://www.wired.com/wired/archive/3.07/dyson_pr.html, last visited: 29/12/00.
- Elkin-Koren N., 'Copyrights in Cyberspace - Rights without Laws?', Symposium on Internet and Legal Theory, Chicago-Kent Law Review, v. 73, n. 4, 1998, pp. 1155 - 1202;
- FAQ About Open Source, Open Source.org, <http://www.opensource.org/faq/html>, last visited: 23/12/00.
- Fisher W., 'Property and Contract on the Internet', Symposium on Internet and Legal Theory, Chicago-Kent Law Review, v. 73, n. 4, 1998, pp. 1203 - 1257.
- Froomkin M, 'A Contract with the Internet', <http://www.law.miami.edu/~froomkin/contract.htm>, last visited 02/07/00.
- Froomkin M, 'An Introduction to the "Governance" of the Internet - Towards an Internet Jurisprudence', 1995, <http://www.law.miami.edu/~froomkin/seminar/ilsx.htm> last visited 15/07/00.
- Froomkin M, 'Of Governments and Governance', <http://www.law.miami.edu/~froomkin/articles/governance.htm>, last visited 12/07/00.
- Froomkin M, 'The Empire Strikes Back', <http://www.law.miami.edu/~froomkin/articles/empire.html>; last visited: 25/03/00.
- Froomkin M, 'The Internet as a Source of Regulatory Arbitrage', <http://www.law.miami.edu/~froomkin/articles/arbitr.htm> last visited: 04/02/01.
- Goldsmith J, 'Against Cyberanarchy', University of Chicago Law Review, 65, 1998, p. 1205.
- Goldstone J, Revolutions: Theoretical, Comparative and Historical Studies' HBJ, 1986.
- Gomulkiewicz R, 'How Copyleft uses License Rights to Succeed in the Open Source Software Revolution and the Implications for Article 2B', Houston Law Review, 36, 179, [1999].

- Greenleaf G, 'An Endnote on Regulating Cyberspace: Architecture vs Law?', University of New South Wales Law Journal, 21, 2, 1998, <<http://www.austlii.edu.au/au/other/unswlj/thematic/1998/vol21no2/greenleaf.html>> (visited 14/07/00).
- 'hacker'(definition); <http://www.tuxedo.org/~esr/faqs/hacker-howto.html>, last visited: 11/01/01.
- Hamerly J, Paquin T & Walton D, 'Freeing the Source: The Story of Mozilla', Open Sources: Voices from the Open Source Revolution, O'Reilly Publishing, 1999.
- Harrett G, 'The Tragedy of the Commons', Science 162, 1243 (1968).
- Harvey D, 'Electronic Civil Disobedience: Universal City Studios v Reimerdes - Liability for Linking and the Provision of Information', September 2000.
- Heller M, 'The Tragedy of the Anti-Commons', Harvard Law Rev. 111, 621 (1998).
- 'Hyper Text Mark Up Language' (definition);<http://webopedia.internet.com/TERM/H/HTML.html>; last visited: 24/01/01.
- Johnson D, 'The Price of Netizenship', November 12, 1996 (draft), <http://www.cli.org/pon.html>, last visited 21/07/00.
- Johnson P, 'Liberal Source Software', Liberal Source Homepage, <http://www.elj.com/lss/lss.html>; last visited: 23/12/01.
- Johnson P, 'The Definition of Liberal Source Software', Liberal Source Homepage, <http://www.elj.com/lss/definition.html>; last visited: 23/12/01.
- Johnson D & Post D, 'And How Shall the Net be Governed? A Meditation on the Relative Virtues of Decentralised, Emergent Law', September 5, 1996, <http://www.cli.org/emdraft.html>; last visited 28/12/00.
- Johnson D & Post D, 'Law and Borders: The Rise of Law in Cyberspace', 48 Stanford Law Review, 1367, (1996), http://www.cli.org/X0025_LNFIN.html, last visited 18/12/00.
- Johnson D & Post D, 'The New Civic Virtue of the Internet - A Complex Systems Model for the Governance of the Internet', <http://www.temple.edu/lawschool/dpost/Newcivicvirtue.html>, last visited 25/07/00.
- Johnson D & Post D, 'The "Unsettled Paradox": The Internet, the State, and the Consent of the Governed'⁵ Indiana Journal of Global Legal Studies.<http://www.temple.edu/lawschool/dpost/Sov.html>, last visited 25/04/00.
- 'Kernel' (definition) <http://foldoc.doc.ic.ac.uk/foldoc/foldoc.cgi?kernel>, last visited: 12/01/01.
- Kaminsky D, 'Core Competencies: Why Open Source is the Optimum Economic Paradigm for Software', Doxpara Research, wysiwyg://20/http:doxpara.netpedia.net/core.html; last visited: 22/12/00.
- Lemley M & McGowan D, Legal Implications of Network Economic Effects, 86 California Law Review, 479 (1998).
- Leonard A, 'How Big Blue Fell for Linux', Salon.com, http://www.salon.com/tech/fsp/2000/09/12/chapter_7_part_one/index.html; last visited: 01/02/01.
- Lessig L, 'Code is Law', <http://cyberlaw.stanford.edu/lessig/content/index.html>; last visited: 04/02/01
- Lessig L, 'Cyberspace's Constitution' Harvard Berkman Centre for Internet and Society, <http://cyber.law.harvard.edu/works/lessig/constitution.pdf>, last visited: 18/01/01.
- Lessig L, 'Governance', CPSR Conference on Internet Governance, October 10, 1998, pp. 1 - 8.
- Lessig L, 'Governance', <http://cyberlaw.stanford.edu/lessig/content/index.html>, last visited: 03/02/01.
- Lessig L, 'Open Code and Open Societies', Keynote Address: Free Software - A Model for Society, Tutzing, Germany, June 2000, <http://cyberlaw.stanford.edu/lessig/content/index.html>, last visited: 04/02/01.
- Lessig L, 'Open Code and Open Societies: Values of Internet Governance' Harvard Berkman Centre for Internet and Society, <http://cyber.law.harvard.edu/works/lessig/opensocd1.pdf>, last visited: 19/01/01.

- Lessig L, 'Reading the Constitution in Cyberspace' Harvard Berkman Centre for Internet and Society, <http://cyber.law.harvard.edu/works/lessig/reading.pdf>, last visited: 27/12/00.
- Lessig, L, 'The Code in Law, and the Law in Code', <http://cyberlaw.stanford.edu/lessig/content/index.html>; last visited: 04/02/01
- Lessig L, 'The Law of the Horse: What Cyber Law Might Teach', <http://cyberlaw.stanford.edu/lessig/content/index.html>; last visited: 04/02/01
- Lessig L, 'The Limits in Open Code: Regulatory Standards and the Future of the Internet', <http://cyberlaw.stanford.edu/lessig/content/index.html>; last visited: 04/02/01.
- Lessig L, 'The Limits of Copyright', The Standard, [wysiwyg://57/http://www.thestandard.com/article/display/0,1151,16071,00.html](http://www.thestandard.com/article/display/0,1151,16071,00.html); last visited: 23/12/00.
- Long R, 'The Libertarian Case Against Intellectual Property Rights', Formulations, Free Nation Foundation, Autumn 1995.
- Loren L, 'The Purpose of Copyright', Open Spaces Quarterly, December 22 2000, <http://www.open-spaces.com/article-v2n1-loren.php>; last visited 29/12/00.
- Lutzker A, 'Primer On The Digital Millennium Copyright Act', <http://www.arl.org/info/frn/copy/primer.html>, last visited: 31/01/01.
- 'Machine code' (definition). Free Online Dictionary of Computing, <http://foldoc.doc.ic.ac.uk/foldoc/foldoc.cgi?machine+language>, last visited: 11/01/01
- 'Mailing List' (definition); <http://www.netaction.org/articles/freesoft.html>, last visited: 19/01/01.
- Markoff J, 'The Concept of Copyright Fights for Internet Survival, May 10, 2000, The New York Times.
- Netcraft Web Server Survey, <http://www.netcraft.com/survey/>, last visited: 19/01/01.
- No Electronic Theft Act; 105 P.L. 147; 111 Stat. 2678; 1997 Enacted H.R. 2265; 105 Enacted H.R. 2265
- Nesson C, Zittrain J & Lessig L, 'Open Code Open Content Open Law - Building A Digital Commons', Strategic Planning Session, Harvard Law School, Berkman Center for Internet and Society, <http://cyber.law.harvard.edu>; last visited: 12/01/01.
- Netscape Communications Corp., *Secure Sockets Layer*, <http://home.netscape.com/security/techbriefs/ssl.html>, last visited: 20/12/00.
- Oliva A, 'The Competitive Advantage of Free Software', 1st Free Software International Forum 2000, May 2000, <http://www.ic.unicamp.br/oliva.html>; last visited: 28/12/00.
- 'Open source software', (definition). Computer User.com High Tech Dictionary, <http://www.computeruser.com/resources/dictionary/>, last visited: 11/01/01.
- 'Operating system' (definition); http://foldoc.doc.ic.ac.uk/foldoc/foldoc.cgi?operating_system, last visited: 12/01/01.
- 'Overview of the GNU Project', Free Software Foundation, <http://www.fsf.org/gnu/gnu-history.html>; last visited: 04/12/00.
- 'Patch' (definition); <http://foldoc.doc.ic.ac.uk/foldoc/foldoc.cgi?query=patch>; last visited: 04/02/01.
- Perens B, 'The Open Source Definition', Open Sources: Voices from the Open Source Revolution, O'Reilly Publishing, 1999.
- 'Pooling Resources', The New York Times, May 10, 2000.
- Post D, 'Chaos Prevailing on Every Continent' Chicago-Kent Law Review, Symposium on the Internet and Legal Theory, v73, n4, 1998, pp.1055 - 1101,
- Post D, 'Governing Cyberspace', 43 Wayne Law Review, 155, (1997), <http://www.temple.edu/lawschool/dpost/governing.html>; last visited 24/07/00.

- Post D, 'Governing Cyberspace or Where is James Madison When We Need Him?', Plugging In, June 1999, <http://www.temple.edu/lawschool/dpost/icann/comment1.html>; last visited 21/11/00.
- Post D, 'Of Black Holes and Decentralised Law Making in Cyberspace', <http://www.temple.edu/lawschool.dpost/blackhole.html>, last visited: 25/03/00.
- Post D, 'What Larry Doesn't Get: A Libertarian Response to *Code and Other Laws of Cyberspace*', January 2000, <http://www.temple.edu/lawschool/dpost/larry.html>; last visited 21/08/00.
- Radin M & Wagner R, 'The Myth of Private Ordering: Rediscovering Legal Realism in Cyberspace', L Bernstein & D G Post (eds) Symposium on the Internet and Legal Theory, Chicago Kent Law Review, 73, 4, 1998, pp. 1295 - 1319.
- 'Rationale for Open Source Definition', OpenSource.org, <http://www.opensource.org/osd-rationale.html>; last visited: 23/12/00.
- Raymond E, 'Critique of the Halloween Documents', 1998, <http://www.opensource.org/halloween/halloween1.html>; last visited: 15/01/01.
- Raymond E, 'Homesteading the Noosphere', <http://www.tuxedo.org/~esr/writings/homesteading/homesteading-6.html>; last visited: 28/12/00.
- Raymond E, The Cathedral and the Bazaar, Eric Raymond's Homepage, <http://www.tuxedo.org/~esr/writings/cathedral-bazaar/>; last visited: 04/02/01.
- Raymond E, 'The Magic Cauldron', Eric Raymond Homepage, June 1999, <http://www.tuxedo.org/~esr/writings/magic-cauldron/>; last visited: 04/02/01.
- Reidenberg J, 'Lex Informatica: The Formulation of Information Policy Rules Through Technology', <http://home.sprynet.com/~reidenberg/reidenberg.html>, last visited: 25/05/00, for the details of such regimes.
- Regulation of Investigatory Powers Act 2000; <http://www.homeoffice.gov.uk/ripa/ripact.htm>; last visited: 04/02/02
- Samuelson P, 'Economic and Constitutional Influences on Copyright Law in the United States', http://www.sims.berkeley.edu/~pam/papers/Sweet&Maxwell_1.html; last visited: 29/12/00.
- Samuelson P, 'Intellectual Property And The Digital Economy: Why The Anti-Circumvention Regulations Need To Be Revised', http://www.sims.berkeley.edu/~pam/papers/Samuelson_IP_dig_eco.htm#ftn20; last visited: 31/01/01;
- Shankland S, 'Linux sales surge past competitors', CNET.com, <http://news.cnet.com/news/0-1003-200-1546430.html?tag=st.ne.1002.bgif?st.ne.fd.gif.j>, last visited: 12/01/01.
- Sonny Bono Copyright Term Extension Act, 1998; S 505, P.L. 105-298, 11 Stat. 2827.
- 'Source code' (definition) Webopedia: Online Computer Dictionary, http://webopedia.internet.com/TERM/s/source_code.html, last visited: 11/01/01.
- Stallman R & Stanco T, 'A Dialogue on Copyright Law and Free / Open Source Software Stallman', <http://linuxtoday.com/mailp...ction=pv<sn=2000-07-13-017-06-NW-CY-SM>; last visited: 07/12/00.
- Stallman R, 'Selling Free Software', Free Software Foundation, <http://www.fsf.org/philosophy/selling.html>, last visited: 20/01/01.
- Stallman R, 'Free Software and Free Manuals', Free Software Foundation, <http://www.fsf.org/gnu/free-doc.html>, last visited: 05/12/00.
- Stallman R, 'The GNU Project', Free Software Foundation, <http://www.fsf.org/gnu/thegnuproject.html>, last visited: 05/12/00.
- Stallman R, 'Selling Free Software', Free Software Foundation, <http://www.fsf.org/gnu/selling.html>, last visited: 05/12/00.
- Stallman R, 'Why Software Should be Free', The GNU Project, <http://www.fsf.org/philosophy/shouldbefree.html>, last visited: 05/12/00.

- Stanco T, 'Birthpangs of a Free World', <http://209.249.55.157/press/etalksF/>; last visited: 05/12/00.
- Stanco T, 'Looking at the General Public License and Open-Source Licenses, Linux Programming, <http://www.linuxprogramming...story.php3?1tsn=2000-07-10-001-03-ID-UU>, last visited: 05/12/00.
- Stanco T, 'Software Licenses and traditional Copyright Law', Linux Programming, <wysiwyg://87/http://www.linuxprogramming...story.php3?1tsn=2000-07-001-03-ID-UU>; last visited 05/12/00.
- Stutz M, 'Applying CopyLeft To Non-Software Information, Free Software Foundation, October 2000, http://www.FreeSoftwareFoundation.org/philosophy/nonsoftware_copyleft.html; last visited: 22/01/01.
- 'TCP/IP' (definition) <http://www.computeruser.com/resources/dictionary/TCP/IP>, last visited: 19/01/01.
- 'The Business Case for Open Source', OpenSource.org, <http://www.opensource.org/for-suit.html>; last visited: 23/12/00.
- 'The Case for Open Source: Hackers Version', OpenSource.org, <http://www.opensource.org/for-hackers.html>; last visited: 23/12/00.
- 'The Customer Case for Open Source', OpenSource.org, <http://www.opensource.org/for-buyers.html>; last visited: 23/12/00.
- The Open Source Definition, v1.7, Open Source Initiative, <http://www.opensource.org/osd.html>; last visited: 22/01/01.
- 'The Open Source Definition', OpenSource.org, <http://www.opensource.org/osd.html>; last visited: 23/12/00.
- 'The Software Gold Rush', Cware, http://www.cwareco.com/gold_rush.html; last visited: 02/01/01.
- 'USENET' (definition); <http://www.netaction.org/articles/freesoft.html>, last visited: 19/01/01.
- Valloppillil V, 'Open Source Software - A (New?) Development Methodology', a.k.a 'The Halloween Documents', I, v1.14, Executive Summary, <http://www.opensource.org/halloween/halloween1.html>, last visited: 23/12/00.
- 'What is Free Software?', The Free Software Foundation, <http://www.fsf.org/philosophy/free-software.html>, last visited: 20/01/01.

APPENDIX 1

GNU General Public License

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc. 59 Temple Place--Suite 330, Boston, MA 02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, 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.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

1. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or 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. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

2. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

3. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

1. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

2. 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.

3. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.) These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

4. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

1. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

2. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

3. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.) The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

5. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

6. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

7. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not

impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

8.If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

9.If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

10.The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

11.If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

12.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.

13.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.

END OF TERMS AND CONDITIONS

APPENDIX 2

The Open Source Definition, Version 1.0

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

1.Free Redistribution

The license may 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 may 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 downloading the source code, without charge, via the Internet. 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.License Must Not Contaminate 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.

10.Example Licenses

The GNU GPL, BSD, X Consortium, and Artistic licenses are examples of licenses that we consider conformant to the Open Source Definition. So is the MPL.

Bruce Perens wrote the first draft of this document as "The Debian Free Software Guidelines," and refined it using the comments of the Debian developers in a month-long email conference in June, 1997. He removed the Debian-specific references from the document to create the "Open Source Definition."