The Open vs Closed Debate

A. A. Adams

pp. 30–47 of 30(3) Journal of Information and Management

Abstract

In business information systems, many of the questions of both ethics and business benefit can be usefully considered in terms of levels and type of openness to be applied. In this paper, I examine both the ethical implications of such choices, but also the business benefits to be gained, from choosing a more open path. software licensing issues, communication protocols, data formats and customer relations are all considered through this lens.

1 Introduction

When developing information systems, whether they be standalone business process support programs for single machines or internet-spanning user-generated content distribution mechanisms, various choices that need to be made in the specification, design and implementation of that system can be characterised as a choice about openness or closure. Sometimes these choices are binary in nature (open or closed, with no in-between), whereas at others it's the level of open-ness that's the question, with a setting available somewhere between fully open and fully closed. In this paper I discuss the implications of various types of choice for various business scenarios, and their relation to general principles of information ethics, such as those espoused by the Association for Computing Machinery (ACM) and the British Computer Society (BCS) in their relevant codes of ethics.

I begin with a discussion of the business issues to be considered in free software versus proprietary licenses and the question of software idea patents, then consider the issue of communication protocols. The related concept of openness in data formats is presented next, followed by the concept of community rather than customers. A brief case study of the issue of openness in anti-malware information finishes the main body of the paper, with the final section deriving conclusions about the benefits or not of openness in these various fields.

2 Code

When the words "open" and "closed" are mentioned to software developers, the first ideas to come to mind are almost certainly the license under which a piece of software will be released: a *free/open source* license or a *proprietary* one. In addition to the other open or closed elements discussed below, this question itself is not so

simple and clear-cut a question as it might appear. In addition to the question of release license, and even this has more to it than a simple consideration, there are also the questions of development environment and the individual rights of members of the development teams, the highly vexing question of software idea patents, and of source code escrow. These questions also need consideration by customers when commissioning software as well as by development organisations.

2.1 Software Licenses

Software licensing is a complicated concept. As is well-documented in, for example, [Williams, 2002], in the early days of computing, software was principally developed by the hardware manufacturers as part of a package of selling expensive hardware to clients, developed for in-house use or developed as part of research by academics. That all changed in the 70s along with the development of personal computers. The growing commoditisation of hardware computing capacity, as always happens with commoditisation, drove down the price of the commoditised good, but also opened up new opportunities for profit in related goods and services. In computing, this new market was principally in programs to run on the machines. By the time Bill Gates was complaining about unauthorised copying [Gates III, 1976] the lines began to be drawn between the proprietary and free software approaches. By 1984, Richard Stallman, already long involved in development of freely shared software such as the Emacs text editor (whose non-legalistic "license" was a social contract and "distributed on a basis of communal sharing, which means that all improvements must be given back to [Stallman] to be incorporated and distributed" [Williams, 2002]) announced the start of a new project: GNU (GNU's not Unix), dedicated to building a free (as in speech) version of the Unix operating system. After much discussion and some soul-searching to find the "sweet spot" for embodying the personal freedom "hacker" [Williams, 2002, Levy, 2001] ideology, the first version of the GNU Public License was produced, setting one of the guiding points of software licensing ever since.

2.1.1 The Scope of a Software License

Since 1976 in the US[1] computer software has been deemed an artistic or literary endeavour attracting copyright in the individual expression (but not the "actual processes or methods embodied in the program", see below on Software Idea Patents 2.3) that embodies that program. It was quickly decided that this extended not only to the source code, which is the actual material written by the programmer, but also to the resulting executable object code. Software is, of course, rather different to other types of written work in that it is useless on its own without hardware on which to run. However, a decent analogy can be made with music which for most people[2] written music is useless without instruments (and for those not musically trained, musicians to play them) on which to execute the sheet music.

Arguments about the legal reach of software licenses have been in progress since the first licenses were issued. When one pays for software, what exactly is one

[1] Following fairly quickly in much of the developed world and more slowly elsewhere.

[2] Terry Pratchett's character Lord Vetinari, who reads sheet music for pleasure without the messy business of musicians getting in the way, aside.

buying? The settled legal view is that software is licensed, not sold. When one "buys" software, one is entering into an agreement whose terms are principally set by the license offered by the owner(s) of the copyright. This license may include restrictions on how many times the software may be installed onto different machines, how many copies may be running at any one time, and certain elements of the usage of the software. However, there are also terms which may not be placed on the purchaser. Users have the right to "decompile" the program for the purpose of "interoperability", that is to figure out how to make other programs communicate with the purchased program. Users generally have the right to sell on a piece of software (in the US, under the "first sale" doctrine [Stefik, 1997] and in the EU under the "exhaustion of rights" doctrine [May, 2003]).

Network Associates Inc. included terms for its Virus Scan software (see section 6 below for anti-malware software as a case study in openness dilemmas) which restricted users' discussion of the capabilities of the program, and in particular its relative benchmark tests. The New York State Attorney General's Office challenged this license as a matter of public policy and won the court case[3] arguing that restrictions such as this were an unconstitutional restriction on freedom of speech, not justified in the protection of trade secrets or goodwill. It was further argued that in particular the ability to discuss the capabilities and vulnerabilities (including comparative benchmarking) of security software was an essential public good that could not be over-ridden by contract terms.

As discussed in [Chandler, 2008, Loren, 2004] the enforceability of "shrinkwrap"[4], "click-wrap"[5] and "browse-wrap"[6] licenses have come under significant scrutiny, as have attempts to restrict who can use software.

Within the non-proprietary software community there are a range of positions on how to license software. Some claim that any restriction on the freedom of others to do what they will with software is wrong. This idea is embedded in the Berkeley Software Development (BSD) license which simply requires that anyone distributing the source and/or object code acknowledge the original authors' copyright and disclaims any liability. Other, such as the Free Software Foundation (FSF), which develops and maintains the GNU General Public License (GPL) regard the share-alike principle (taking someone else's work, and adding a small amount to it should not allow you to deny similar rights to your users) as paramount to maintaining a free information infrastructure. Contrary to the belief of some, the GPL does not constrain programmers to distribute their amendments, not does it prohibit charging for providing the amended code. In practice, however, anyone who does charge could easily find their business model undermined by a single paying customer who chooses to then pass the software along without charging a fee. The GPL *does* require that the amender gives the redistribution and derived work production rights

[3] Spitzer v. Network Associates, Inc. dba McAfee Software 758 N.Y.S.2d 466 (Supreme Court N.Y. 2003).

[4] Software sold in a cellophane-wrapped box, the details of the license being inside the box and stating or implying that opening the cellophane wrapping constitutes agreement to the license.

[5] A software license whose terms are presented to the user when they attempt to install the software. Typically, the user must select an "I agree" box or similar in order for the installation to proceed.

[6] A software license approach legally discredited in the US by Specht v. Netscape No. 01-7860 (L) (2d Cir., October 1, 2002). A license appears somewhere on a web page which also includes a link to download the software thus licensed.

to anyone to whom they distribute the software. There are those who argue that the GPL is not a "free" enough license, as mentioned above, and that by imposing the share-alike principle they are infringing on others' rights. There are others who argue that the GPL is no restrictive enough, but still in the name of "freedom". Thus there are licenses such as the "Hacktivismo Enhanced-Source Software License Agreement" which attempts to prohibit governments (in particular) from distributing amended versions of their software which have been altered to allow government spying on citizens' computer usage (see www.hacktivismo.com/about/hessla.php).

2.1.2 The Ethics and Business of Choosing A License

The example of Microsoft (MS), which started complaining about unauthorised copying[7] of their software in 1976 and today are one of the world's largest companies, would seem to dictate that the only sensible business decision to make when writing software is to use as strong a proprietary license as one can, and even to reach as far into restricting competition as to limit the free speech of users with respect to discussing any possible failings of the software with others. However, as presented below, even for commercial operations and sound financial reasons, a strong proprietary license may not be the best choice even for commercial software producers. For individuals and organisations commissioning software, or having software written in-house and/or by contractors the issue is even more blurred.

In some ways it is unfortunate that the unpaid community effort element of the Hacker Ethic[Raymond, 2001, Loren, 2004] has come to represent free software as a concept so strongly. The volunteer hacker working on software in their free time, donating their time and expertise for the altruistic good of the community, the thrill of producing elegant code[8] and the egoboo[9] from public recognition of one's efforts and skills has become not just the poster-child for free software but the only image many have of free software developers. However, a significant amount of free software development is done by people during work time in ordinary paid computing jobs and while a small proportion of this may be regarded as a "charitable contribution" by the organisation, it is more often the organisational version of "scratching one's own itch" than anything else. Other companies make free software pay sufficiently well to fund its development in consultancy, bespoke development, or support contracts.

Where software is developed in-house, many managers see the development and maintenance of this software as a costs centre only, and may seek to sell the software. For many reasons this is almost always a mistake, since developing software is a risky

[7] Piracy is a hideous crime involving the armed hijack of a vessel at sea and the theft of the cargo and even the vessel itself often while employing savage violence against the crew and any passengers. The link between this heinous crime and unauthorised copying of material under copyright is one drawn by those who cannot justify the status of their holding of copyright, and who must label the act of copying as something so hideous that no one can argue against it as the only way of winning their case. The author is with Richard Stallman on this usage and therefore describes it technically as unauthorised copying. Even the "illegality" of such an action is uncertain in many cases without deep examination of the circumstances — not all unauthorised copying is illegal.

[8] or, in unfortunately too many cases code which the hacker feels is elegant but which is in fact uncommented unmaintainable spaghetti code

[9] A term from science fiction fandom derived from "ego boost" to describe the pleasure gained from recognition for voluntary works. The prevalence of science fiction fans in early free software and online communities transferred the term (and provided some of the impetus behind community efforts as the driving force) to the free software community.

business venture that non-specialists usually fail at, and in addition the software may well represent (part of) the company's principal business advantage in it's principal field of operation. However, where the organisation in question is not a competitive commercial player, but a public sector or non-profit organisation, then the commercial advantage argument turns on its head. Since such organisations by their nature *should* (though for various political reasons they do not always see this) be interested in "raising the game" of others cooperatively. This is particularly true in, for example, local government. In the UK, there has long been a tradition of sharing in-house software between authorities. While often not released beyond the "club" of UK local government, it is effectively a free software distribution model within a closed group of separate entities.

So, if one is working within the public/non-profit sector, there is a strong ethical argument that in-house software developments should be released under some form of communal access agreement. Indeed, if there is potential for a broader utilisation of such software beyond the relevant type of organisation, then a strong argument can be made that there is an ethical duty to release the software more widely. This argument has strengthened over the last fifteen years (such software products and sharing have existed long before then) with the development of effectively free systems for distribution of such software, for example via the Free Software Foundation's website or the Sourceforge site. As we discuss below in section 4, similar arguments can be made regarding the ethical argument for public sector organisations to use open data formats and to make some of the data they generate freely available in such formats.

Even for commercial companies with specific needs, developing in-house software under a free software license may be a sensible business decision. The adoption of a free software license approach allows the development team to incorporate elements of existing free software as part of their system. In-house systems teams need to be aware of the necessity to ensure that management, particularly management outside the technical department, are aware of the decision to use free software and need to keep good documentation on the origins of their code. In particular, it is incumbent upon IT staff to ensure that any suggestion of taking in-house developed systems and distributing them is provided with clear guidelines on the status of the inherited code and the implications of its license for any such distribution.

2.1.3 Commissioning Software

So far we have focussed on the license adopted by an in-house team developing software for internal use. However, many businesses use software company for their bespoke IT needs. Questions of licensing on the side of the commissioning company need careful consideration, which they often do not get. Many software houses will take a line on licensing that tries to tie in the commissioning company to the software house for future development work, and retains ownership of the commissioned work by the software house, for possible re-use in other projects, and even development into a package for general sale. Again, the commissioning company need to consider their needs in the negotiations, and need to consider the future carefully. Larger companies and public sector organisation, such as government departments, where they commission software from a third party will often, though not always, take the longer-term view and insist on some form of access to the source code. This can

take a number of approaches, but can include transfer of ownership in the software, shared rights, source code escrow and code audit rights.

2.1.4 Transfer of Ownership

A software house will typically demand higher payment for software developed with a transfer of ownership compared to software in which the company retains some or all of the rights. In fact, it may not be possible to transfer all of the rights to the commissioning organisation. The software house may have developed a set of libraries for various purposes some of which may already be used in previous projects, and the use of which for future projects may be absolutely necessary for their continued business. If the company has based some of their work on GPL or LGPL[10] then they may not own the rights themselves to transfer. If a software house has the right skills to develop the software that the commissioning organisation needs, then it is likely that a full transfer of rights would not be sensible for the software house.

For the commissioning company, sole ownership is the equivalent of developing the software in-house, without needing to employ the software developers directly or on a long-term basis. It provides the benefit that the software can be later further developed in-house or by a third party, either for in-house use or for sale as a product, a strategy already dismissed above.

2.1.5 Shared Ownership

Shared ownership is a shorthand for a number of different ways of licensing the output of a piece of bespoke software. At one extreme the commissioning and developing organisations have equal but separate rights to the output of the project at the end. Each is free to develop the code further on their own or with other parties, and neither is constrained as to how these further development may be commercialised or used. At the other extreme, the commissioning company may have full access to the source code and may use it and develop it for in-house use themselves or bring in contractors to perform such development, but may not sell the software to others nor pass on rights to third parties. The most common variant of this kind of arrangement lies in the middle, with pre-existing libraries etc from the software house being licensed for use by the commissioning company, perhaps including a time- or type-limited right of access to updates (such as updates to new versions of the operating system). The new software developed during the project is then subject to the common ownership of the commissioning and developing organisations as detailed above.

2.1.6 Source Code Escrow

Small software houses, like all small businesses, are in a precarious position. Without any ill intent on behalf of the owners and management of such companies, they

[10]The GNU Lesser General Public License which allows other programs to call the relevant software as a separate library without the "share-alike" element of the GPL coming into play for the core program, only for any changes to the library.

frequently fold and their records, including their source code base, may be deleted or may end up in the hands of liquidators for whom their sole purpose is the realisation of the maximum funds from those assets in as short a time as possible. When one has commissioned software from a company which then goes under, the possibility is that both the original source code and/or the clear chain of ownership and associated rights, is lost. To defend against the negative impacts of this, but where the software house is unwilling to agree to a transfer of rights or shared ownership, the source code may be placed in "escrow": that is, it may be lodged with a trusted third party. A properly drawn up escrow agreement provides the commissioning company with both access to the source code and the equivalent rights of shared or transferred ownership if and only if the software house ceases trading, or possibly on violation of the contract. This ensures that the commissioning organisation is not left with a set of running object code but no way of fixing bugs or upgrading it. The trusted escrow company ensures that the original software house's rights are not voided until and unless they are out of business. Alternatively a contract providing the code to the commissioner but no rights to do anything, except possibly read it, can be agreed.

2.1.7 Code Audit Rights

The right to at least see the code of a commissioned piece of software, or even one which is available off-the-shelf, may be a requirement for certain types of business. Partly in response to concerns about deliberate backdoors placed in the dominant MS Windows operating system from various non-US governments, and in reaction to other pressures such as continuous accusations of monopoly leveraging from operating system to office software and back (see also section 4 on data formats) MS introduced the "Shared Source Initiative" which provides users of, and developers for, some of MS's products access to (part of) the source code of systems such as MS Windows and MS Office. Similar code auditing arrangements can be built into bespoke software development contracts, allowing representatives of the commissioner, usually including contractors, the right to check the source code for security holes, whether deliberate or inadvertent.

2.2 Open and Closed Development Environments

The output of a software project and the tools used to develop it have no technical requirement to share license structures. For example, the use for a number of years of a proprietary tool (BitKeeper) for controlling the source code of one of the most high profile free/open source programs (the Linux Kernel, a vital part of the GNU/Linux operating system) was very controversial. A license move by BitMover (the company who produces BitKeeper) in 2002 [Shaikh and Cornford, 2003] which tried to restrict use of BitKeeper and prevent developers of interoperable free software clients:

> this License is not available to You if You... develop, produce, sell, and/or resell a product which contains substantially similar capabilities of the BitKeeper Software, or, in the reasonable opinion of BitMover, competes with the BitKeeper Software .

caused a significant disagreement amongst Linux developers. The controversy was somewhat lessened when BitMover released their own free software (GPL v.2) client called bk-client to connect to BitKeeper repositories, although with limited capabilities. The controversy was re-ignited in 2005 when BitKeeper withdrew their support due to the efforts of some Linux developers to develop a full-featured client to connect to the BitKeeper servers. As a result even Linus Torvalds finally shifted his stance and engaged with other Linux developers to scratch their own source control itch and produced the free software system Git, a source control system designed for very large highly active projects.

2.3 Software Idea Patents

In 1979 the spreadsheet, often claimed as the *killer app* that ensured the success of the personal computer in ordinary office environments, was invented by Bricklin and Frankston. They did not attempt to file a patent, although they did consider it (a common myth is that patenting was not seriously considered by any programmers in the 70s). Bricklin describes their consideration on his personal website www.bricklin.com/patenting.htm:

> Why didn't we patent the spreadsheet? Were we stupid?
>
> This is a very common question, since, by the late 1990's, software inventions were routinely patented. Today, it seems negligent to ignore patents. However, in 1979, when VisiCalc was shown to the public for the first time, patents for software inventions were infrequently granted. ... The publishers of VisiCalc ... retained a patent attorney who met with executives from Software Arts and Personal Software. The patent attorney explained to us the difficulty of obtaining a patent on software, and estimated a 10% chance of success, even using various techniques for hiding the fact that it was really software (such as proposing it as a machine). Given such advice, and the costs involved, we decided not to pursue a patent. ...

By 1981 the situation had changed and software patents were being granted, although this growth of the patent system has been far from universal, and both the concept and its implementation in the US today remain controversial. The inclusion of software purely as software (and not as part of a larger invention) in the scope of the patent system outside the US has been one of the few failures of the US' commercial rules hegemony on so-called *intellectual property* rights [Drahos and Braithwaite, 2002] over the past thirty years. At present, despite various attempts by the European Patent Office[11] (EPO), various major international firms who hold software patents in the US and would like to see them introduced worldwide, and the US trade representatives [Drahos and Braithwaite, 2002], so far the European Parliament has resisted attempts to introduce software patents. It is a perennial issue, however, and as of writing yet another discussion of European

[11] It is alleged that as well as arguing on a policy-making level that software should be patentable in the EU as it is in the US, it is often alleged that the EPO has deliberately stretched the rules, or even broken them, in granting patents on software in an attempt to by pass the democratic decision-making of both the EU and Member States and effectively introduce software patents by grant rather than by statute.

patent reform is underway, with various groups offering suggested wordings which they all claim will simply clarify matters and reinforce the status quo. Of course if one believes the allegations about the EPO then that status quo includes the granting of patents on software in the EU.

Stallman, and other free software exponents (including those on the *open source* side of the free/open source software philosophy) are vehemently opposed to *software idea patents*. As they point out, software is inherently abstract except where it controls a physical entity. Restricting the use of software concepts, it is claimed, undermines not only the whole ethos of free software[12] but would also undermine the rest of the software industry, producing such ridiculous barriers to entry that few others than today's software giants (MS, IBM, Oracle and Sun, all of whom hold vast cross-licensed patent portfolios) could ever afford to produce software. Critics of software idea patents also point to the fact that innovation in software has been, and continues to be, incredibly healthy, even without the 'protection' supposedly offered by patents.

3 Communication Protocols

In defining communication protocols, there are a number of issues where the question can be classified as open or closed. These are: trust, interoperability and interface.

3.1 Trust in Communication Protocols

When the internet was first developed, certainty of delivery over uncertain physical connections was the principle at work in defining the protocols for communications. All computers attached to the network were *trusted* not to be the source of malicious communications. Thus when the protocols of the first individual to individual communications tool (email) was defined (RFC 821: SMTP [Simple Mail Transfer Protocol] in August 1982, Jon Postel) the system was designed to include the minimum overhead of information exchanged to enable transfer of the important content. As such, SMTP is a highly trusting and open protocol. A receiving machine trusts the data provided by the sending machine in terms of the origination of the data and by agreeing to forward messages to other recipient machines. As the internet expanded, this trusting approach could not be maintained and many systems closed down their willingness to forward messages to other recipients, unless the message comes from a known source. The volume of spam long ago reached the point that accepting mail from everywhere for forwarding is now not only discouraged but such *open mail proxies* are now routinely blocked from sending messages to most of the rest of the internet.

However, even now, reverse lookup of domain names compared to internet addresses are deemed too expensive, and more importantly too restrictive, the be implemented on many mail receipt systems. Email is, in many ways, the victim of its own success and constantly teetering on the brink of being overwhelmed by

[12] which has a license which cannot prevent distribution free of charge, and which therefore it would be difficult to find a way to provide the funding for patent license fees, let alone actually track distribution to assess per-user fees, the usual basis for a patent license

malicious use. Being a relatively light overhead (even now) communication protocol, and given the openness of its implementation even now, a large number of emails are program-to-person or even program-to-program communications. This provides one of a number of barriers to a wholesale replacement of the SMTP-based email system still in use today. The other barriers to its replacements include:

- The immense install base of email: every user online has at least one email account and more usually more than that. Some will run hundreds of accounts on different systems for different purposes.
- The criticality of email in person-person machine-person and machine-machine communications, and in particular the lack of knowledge of many systems people of exactly which protocols some of their software uses.
- The desire of governments to censor and monitor their citizens' communications produces in some circumstances a pressure from surveillance authorities to maintain a relatively-simple-to-eavesdrop-on service, while suggestions for authority-assigned trackable origination communication are resisted by citizens wary of the hidden agenda of surveillance cloaked in rhetoric about reducing malicious communications.
- The suggestions by some major players in the software industry to include proprietary formats, protocols and even patented methods in revised standards.

3.2 Communication Protocols and Interoperability

Communication is all about connecting people with each other, preferably when both of them wish to be connected. In a small network this can be handled by a variety of means, but the numeric rules of networks mean that beyond a relatively small size of network, scalability demands distribution of at least some authority. In addition interoperability becomes a major issue in any definition of a communication protocol. Take the example of instant messaging (IM). The concept of direct synchronous communication between users has been a feature of computer communications for a long time, the specification for IRC (internet Relay Chat: RFC 1459 May 1993 by Oikarinen and Reed) provided an open protocol by which users could log on to a server, select one or more channels of communication on that server, and converse with each other with a minimum of typing overhead. This open protocol served traditional internet user for many years and became one of the main communication alternatives, particularly for synchronous groups discussions, along with email and usenet. However, for the non-expert computer user AOL, Yahoo! and MS came to dominate this communication space.

The walled garden online service of America OnLine (AOL) introduced AIM (AOL Instant Messenger) in 1996. In 1997 they released a client for non-AOL internet users, but still only AIM-registered users. In 1998 Yahoo! introduced their "Yahoo! Pager" system, including the familiar *buddy list* feature. Again, this system included a single log-on to the Yahoo! central server and one could only talk to other Yahoo! users. In 1999, MS released MSN Messenger as part of its MS Network (MSN) online offering. The original MSN Messenger client allowed users to interoperate with both AOL and Yahoo! IM systems, by providing their account details to the Messenger server. AOL, in particular, strongly criticised this practice

[Hu and Junnarkar, 1999] citing security concerns but almost certainly more worried about their market share and the advertising revenue generated by the ad-supported standalone client as well as their own subscriber base. As we discuss below, AOL have also attempted to block other clients from connecting to their messaging server (claimed to be in an attempt to secure the system, but in reality almost certainly in defense of their revenue stream). After regular alterations to the server and message protocol made keeping changes up to date on the Messenger client impractical, MS abandoned attempts to technologically force interoperability of IM on AOL.

For many years after their launch, each of these systems stood alone. Unlike email which, despite efforts by many companies to appropriate it to their own private formats remains mainly open as discussed above, these separate systems came to dominate the instant messaging were restricted to communication with other users on the same system. Many users maintained accounts on more than one, sometimes all three of these different systems. In 2006, MS and Yahoo! launched interoperability between their services. AOL, having been approached to join them, still maintains its standalone network.

Despite its open definition and multitude of servers and clients, the Jabber protocol remains a small player in the IM field. Newer services and systems such as Facebook and the iPhone continue to challenge AOL, MS and Yahoo!'s dominance, and internal instant messaging within organisations exist, often based on proprietary systems such as the venerable Lotus Notes internal communication suite.

4 Data Formats

Closely linked to the issue of communication protocols are the issue of data standards. Everything from SMS (text messages) limited to 56 characters up to the entire contents of the internet archive (formerly the wayback machine) which aims to preserve the contents and changes of the internet for the future, is stored in data formats of varying levels of abstraction and complexity. At base, digital data is simply a collection of ones and zeroes. It is the interpretation of that data that makes it meaningful and useful. In the early days of digital electronic computing, even the interpretation of the ordering of the ones and zeroes as a binary number was a source of format incompatibility, the "bid-endian" versus "little-endian" approach. Even now, different computer hardware runs on different endianness due to a variety of historical and purpose-optimisation reasons beyond the scope of this article. Digital information is valueless if its format is not understood. This may be a deliberate part of a format, in fact, for example formatting information stored in marked meta-data tags may be ignored by devices or programs for whom that data is not relevant. This allows the possibility of backwards compatibility, an important element in data formats since for many applications a universal simultaneous upgrade can not be expected.

Data formats can be classified into three main categories: completely closed; published but proprietary-controlled; open standards. Examples of each of these formats include:
Closed: The MS Office formats. Despite a badly flawed process allowing the specification of MS's XML-wrapped binary formats as a "standard" MS's Office formats, including .doc[x], .xls[x] etc. remain closed proprietary formats. Reverse engineering

allows some interoperability of other software, such as OpenOffice.org, but this is far from perfect due to a lack of published specification for the meaning of some of the data.
Published Proprietary: The portable document format (PDF) developed and maintained by Adobe Inc. The full language specification is published by Adobe with the intent that programs other than those produced by Adobe can interpret and produce well-defined PDF files. Changes to the format are controlled by Adobe.
Open Standards: Despite early tag proliferation in browsers and web page editors, HTML (and related formats such as CSS, XHTML etc.) is now a standard developed and published by the World Wide Web Consortium (W3C). The drafts are developed by the W3C's HTML Working Group.

As described above with regards to software licensing, choices regarding data formats are often not given the attention they deserve in companies. In particular, companies choosing the software they will use to run their business often ignore the question of the format of the data they will hold. That data is crucial to their business, and a requirement to re-produce data from scratch due to lack of current software to interpret older data can be sufficient to devalue or even close a business. So, when considering software and data formats, businesses need to look both forward and backwards.

4.0.1 Data Formats for Software Houses

The meaning of a data format is open to anyone with access to the source code of a program that can interpret that data. While not the most efficient or simple of ways to provide access to a specification, it does the job. So, if a software house produces free software, or allows access to its source code under more or fewer conditions, then the data format specification is discoverable by those with that access.

If one is producing a proprietary program then one may still produce data formats following a published description (whether proprietary or open standard). Adobe's Acrobat and MS's Internet Explorer are both proprietary programs using published data formats.

Some customers, particularly public sector customers, have as part of their requirements that data produced by programs is stored in a specified format (see below on good practice for software users).

The reason for such requirements is the loss of data that has already occurred in many organisations, particularly large public organisations who invested early in the creation of large amounts of data. When a software or hardware company went out of business, unless the information technologists working for the organisation were sufficiently on the ball, the capacity to use existing software and hardware to produce data usable by the new systems could easily be lost [Digital Preservation Coalition,].

It might seem an obvious benefit to allow a program to read in as many external formats as possible, whether proprietary (one's own or the reverse-engineered versions of other's), published or open standards. However, the brings one's own programs at least partly into the interoperability region. While reading data format X produced by program Y, a competitor to one's own offering Z, might provide a

relatively simple transition route from program Y, it also allows for the possibility of companies maintaining program Y as a main option and using a very small number of licenses for Z where it capabilities are really needed.

Output formats are the trickier question, however. Vendor lock-in using proprietary formats has been the model of commercial software for almost three decades. The examples are numerous, with reverse-engineering of the basic data format battling against deliberate obfuscation of the meaning of the data by the original firm. As mentioned above, however, there are other options. In order to avoid losing control to a standards body, which may move slower than user needs, and may be subject to capture by one's opponents and used to prevent new capabilities from reaching the market in time to secure a solid commercial advantage, the Adobe option of publishing the data standard but retaining control of it, seems very sensible. Of course, this may still be captured by competitors branching the specification for their own purposes ("embrace, extend extinguish").

If one does not already have a lead in the marketplace, gaining it can be very difficult [Brynjolfsson and Kemerer, 1996]. In this case, subscribing to standards bodies, gaining a seat at the table, and competing on the merits of one's programmers and vision of user needs, may be the better option. Certainly a revolt by users who may see the proprietary data format as a stranglehold on their own valuable data, and upgrade or maintenance costs that rise over time as a sufficient reason to bite the bullet and change platform (likely never to return) are a risk of the proprietary route.

4.0.2 Data Formats for Other Organisations

For organisations commissioning bespoke software/hardware or buying off-the-shelf materials, the choice of data formats which are in some sense open to them really should be the obvious choice. A good range of options for both input and output formats, and adherence to data format standards, are all good for their business. For the public sector in particular, this issue has been gaining some political traction for some time. The difficulty comes, as always, in the interoperability stakes when other organisations work with a closed standard. In the UK for example, some government contract documents have in the past been issued solely in MS Word format, and required the advanced scripting facilities of Word to be completed. Not running a professional edition of MS Word would bar an organisation from bidding for government tenders. Direct online submission systems are tending to move away from this desktop program model, but it can still cause problems. Digital access and preservation strategies need careful thought alongside interoperability questions, and the possibility of loss of everything must be weighed against the possible loss of richness in some data.

5 Community or Customers

In these days of user generated content the phrase *community* is often bandied around by those running websites where the provider is solely an intermediary between users sharing their material. There are many analyses of the economics and social norms attaching to this idea [Surowiecki, 2004, Tapscott and Williams, 2006,

Constitution Committee of the House of Lords, 2008, Shirky, 2008]. Here we consider the attitude of organisations to their main users. Those organisations can be etail (Amazon.com), information provision (Wikipedia.org, LATimes.com, YouTube.com), personal introductions between users (adultfriendfinder.com) social networking sites (Facebook.com, Mixi.jp) and many others. One of the things each of these sites has in common is that part of what they provide is generated by their users. In some cases, e.g. Wikipedia, it is all they provide. At the other end of the scale the LA Times online site is principally populated with the organisations' own material with some commentary and discussion amongst users. From Amazon's customer ratings and reviews to adultfriendfinder's profiles to Facebook and Mixi's blogs, photos and friend lists, part of the buzz about these sites is their *community* rather than their *customers*. The software industry, combining their own and their users' technical expertise, were one of the first types of business to see the advantage in helping their users to communicate with one another. However, this can be a double-edged sword. Whereas harnessing the Hacker Ethic [Himanen, 2001] of information sharing provided free *user support* by creating a resource of experienced users willing to share their knowledge with each other (and, crucially, with new users) it also creates a perfect opportunity for unhappy users to report their woes to other users and potential users. Of course, these days most people have ample other places to air their poor customer experiences, but few have the fame of Eugene Volokh [Solove, 2007, p. 93] and hence a blog post by most will barely attract any significant attention, but if it's more than a few complaining bitterly on an official site, then the organisation can have real problems. When posted to an organisation-owned *community* site, such comments can, perhaps, be removed, but the very act of creating a forum where users provide most of the content is that they feel that they gain some, if not all, of the expected free speech rights of public places [Sunstein, 2002, Klein, 2000] and hard censorship on such sites will tend to undermine their benefits in general while possibly driving away existing customers more unhappy with the censorship than because of the original failings.

Where the content of a site is created principally by the users, and the organisation provides a structure in which to hold that content, questions of ownership come into play. The rhetoric of publishers has long been that copyright derives from the brilliant acts of creation performed by the artist (writer, composer, musician) and that the long, strong and wide protection is needed to provide the deserving creator with the just benefits. of course, the fact that the middlemen have been taking the lion's share of all the income is never mentioned. This case becomes even more difficult when sites such as YouTube are considered. Created in 2005, YouTube quickly gathered a huge base of video clips from users, free for other users to view. These clips included home videos (everything from the funny antics of a cat to political diatribes), film student shorts and extracts from commercial material. Even in the early days the commercial content caused some difficulties for the site operators, with complaints from music recording companies, film companies and television production companies. When Google acquired YouTube for in 2006 these companies redoubled their efforts to either shut YouTube down, or at the very least acquire a share in its profits. This raises an interesting question, however, of what Google actually bought, and from whom. The terms and conditions of the YouTube site are that uploaders provide an irrevocable license for YouTube to do pretty much everything they want with the material thus uploaded. In typical middleman style this included selling the entire site to Google for $1.65b without

ever paying existing users a cent for their content. Under various pressures, Google introduced a profit sharing plan in 2007 to enable uploaders of very popular videos to receive some of the advertising income that funds the service. It remains one of the few user-generated content sites to have any form of plan. No such site to my knowledge has shared an IPO or massive sale proceeds with their users.

6 The Anti-Malware Prisoners' Dilemma

Anti-virus software has been around for a long time. Viri and other forms of malware have been circulating since the early days of the mini computer and the forerunner of the internet. Anti-virus and related security software has become big business. Major community efforts are also put into identifying both vulnerabilities and threats. CERTs (computer emergency readiness/response teams) exist in all the developed countries and a number of others, with local teams also in existence. The threat and risks are seen as so important that the main US-CERT is now part of the Department of Homeland Security.

Like much else in the world of security, most people are not very good at dealing with it. Many do not install any significant protections until they have been badly damaged at least once, and sometimes a number of times, by malware attacks. Given the potential for indirect harm from many modern malware, where the machine in question is merely hijacked (turned into a *zombie*) and its network connection used for the sending of spam, participation of distributed denial of service attacks or for use in cracking high-value machines, this is an unfortunate position. It does mean, however, that advertising and selling security software is difficult. Competition in this marketplace is therefore quite fierce. This is one area, however, where competition probably does not produce the optimum outcome. The reason for this is that there are two major parts to anti-malware programs. The first is an overseer program that monitors what is going on in the operating system and looks for other pieces of software whose actions are out of the ordinary (accessing many different types of file, adding the same code to each, for example). The second is a set of recognition signatures for identified malware and unpatched vulnerabilities in programs. If a piece of malware appears on the computer via any route the easiest, quickest and most effective way for the anti-malware program to identify it is with a software signature: some element(s) of the virus' code identified as unchanging between minor variations.

In an ideal world these virus signatures would be freely shared between anti-malware companies and given away free to users. However, identification of these signatures is a time-consuming job and while some of this is done by volunteers or CERT team members, much of it is done by paid employees of the anti-malware company. While free sharing of the signature files would probably make all computers safer from infection, such sharing would also, particularly if not reciprocated, reduce the competitiveness of a company's software in the security marketplace. If Company A shares its identified signatures with everyone else, then the software from all companies will detect the viri that Company A's software does, plus the ones they have separately identified but not shared. In addition, it is likely that *churn*[13] in the marketplace is at least partly driven by failure of the existing soft-

[13] That is, customers shifting from one product to another.

ware to prevent infection. So, sharing virus signatures can reduce churn, and thus reduce the ability of companies offering better programs to gain customers during the *churn*. So, instead of *cooperating*, the anti-malware companies *defect* and everyone loses out.

7 Conclusions

When software is produced by public or non-profit organisations, the mode of operation should almost certainly be some form of free software license. Not only does this allow their programmers free reign to build on existing free software and not re-invent the wheel, but by explicitly collaborating with other public/non-profit organisations with similar problems to solve, a cooperative effort can yield quicker and sturdier results than a solo effort in each organisation. The (ideally) lack of any form of competition among public and non-profit organisations generally removes any doubts that internal developments should be released in this way.

Similarly to public and non-profit organisations, commercial organisations should consider building on existing free software projects, either forking them to develop into serving their own needs or contributing to the general development of a particular project. Few pieces of software developed for in-house use are ever successfully commercialised and building internal tools on free software can prevent managers from following this dangerous path.

As has been shown by a number off companies, some of them major players (IBM, RedHat, Sun) developing free software can provide valuable income. Instead of competing with other companies for a locked-in client base paying for regular upgrades, maintenance and support of a proprietary program, competing for business offering a specific service can lead to a healthier long-term business. Certainly the ethical values of the ACM and BCS where the customer, society, the developers and the business are all seen as stakeholders in the information infrastructure, free software is much more compatible with these values. The use of free software as a company's offering encourages it to regard its highly skilled development staff as its most valuable resource, and not as a cost base whose benefits and salaries should be kept down at all costs. Treating one's knowledge workers as the core of the business fits well with the shift to the knowledge economy described by so many including [Castells, 1996, Castells, 1997, Castells, 2000]. Regarding one's knowledge workers as less important than the managers and sales staff is the stuff of Dilbert's world and can often be the downfall of a company, in the long if not the short term.

The arguments against patents generally have been made many times for both philosophical [Jefferson, 1907, pp.326–338, Vol. XIII, Letter to Isaac McPherson] and practical [Drahos and Braithwaite, 2002] reasons. The particular case against restrictions on software are compelling:

- Without software patenting, the software industry in Europe has not imploded nor suffered from a lack of investment.
- The level of innovation in software is very high and as patents are supposed to promote innovation by rewarding it. Since the innovation exists patents are unnecessary.

- Few software patents issued in the US have ever been successfully defended against charges of lack of novelty, obviousness (to a relevant practitioner), or published prior art.
- If the idea of the spreadsheet had been patented in 1979, the base idea would only have dropped out of patenting in 1999. The world of computing in 1999 and that of 1979 were so radically different it is hard to see this as a sensible term.
- The return on investment period for software is so variable, depending on too many factors, for a sensible term to be valid across software.
- Software is so abstract, malleable and variable that the only beneficiaries in the long run are likely to be lawyers.
- Even for large software concerns, the risks in developing any new code would be substantial whereas the rewards are uncertain.

Publishing the details of communication protocols usually benefits the developers of the original system using the protocol because the network multiplication factor outweighs the detriments of increased competition. Sometimes a first mover can gain and maintain a closed protocol system, but they run the risk of losing their market share very quickly (e.g. the closed protocols of Friendster, one of the early mass appeal social networking sites lost out to the similarly closed protocols of Facebook, although the closed protocol of AIM still puts AOL at the top of the IM market). Protocols themselves should not be too open or trusting of information coming in. The early days of trusting everyone on the network are long gone and there is too much pollution in the information stream for open, trusting protocols.

Where feasible, the users of software (whether that be off-the-shelf or bespoke) always benefit from known formats. The differences between published but owned formats and true standards are complicated and often depend on the particular type of document, its purposes and user base. Proprietary formats, like closed protocols, can produce customer lock-in and dominant market share, but reverse-engineering is likely to cause an arms race and some large customers such as governments may well have the power to demand, or even force, the opening of a format specification.

Members of a community are more likely to be forgiving of failings than pure customers, and to have some sort of emotional investment in the software or service they are using. However, members of a community also have greater expectations of responsiveness and other elements of human rights come in to play. The ACM and BCS code of ethics, with their stress on balancing stakeholder interests including those of customers should push computing professionals down the road of community-building. Provided you treat your customers well and don't set out to exploit them. the benefits can far outweigh the downsides, particularly in user to user support.

7.1 The Costs and Benefits of Open-ness

Being open has its risks, particularly for things like communication protocols, where trusting protocols are now almost always abused. By adopting an open approach in

general, businesses can develop a more balanced approach to their activities, being paid for what they do and will do, rather than trying to be paid for what they have already done. Higher risks may bring higher rewards in the long run, but they also may not. If long term value is your goal, a more open approach can produce a more efficient economy, which can have broad benefits. In particular, the use of free software licenses, open formats and building communities should help us to develop a more robust software infrastructure. Competition based on continuing ability to meet user needs is healthier than one based on long-term lock-in, see-saw (teeter-totter) economics [Hunt, 2000] and customer exploitation. Arguments about universal interoperability are often used by companies such as Microsoft to explain their effective-monopoly position as good for the consumer. Open formats, clear distinctions between infrastructure, utility and productivity applications and competition in the marketplace reduce barriers to entry, and force companies to maintain the quality of their offerings far more than a captive market. Excess profits in the software industry syphon money from elsewhere in the economy and claims that this is healthy are an example of the Broken Window Fallacy [Bastiat, 1850]. Monopolies can also be brittle in a number of ways, including security risks associated with monocultures (if almost everyone runs a particular piece of software then any vulnerability in that software makes it more attractive to attackers and more devastating when an attack occurs), and the problem of one monopoly replacing another, whereby the previous monopolist may swiftly find themselves bankrupt.

References

[Bastiat, 1850] Bastiat, F. (1850). What is Seen and What is Not Seen. In [Bastiat, 1964]. Original essay published in 1850; available online.

[Bastiat, 1964] Bastiat, F. (1964). *Selected Essays on Political Economy*. Van Nostrand, Princeton, NJ.

[Brynjolfsson and Kemerer, 1996] Brynjolfsson, E. and Kemerer, C. F. (1996). network externalities in microcomputer software: An econometric analysis of the spreadsheet market. *Management Science*, 42(12):1627–1647. www.katzis.org/wiki/images/9/9f/Brynjolfsson_1996.pdf.

[Castells, 1996] Castells, M. (1996). *The Rise of the Network Society*. Number 1 in The Information Age. Blackwell, Chichester.

[Castells, 1997] Castells, M. (1997). *The Power of Identity*. Number 2 in The Information Age. Blackwell, Chichester.

[Castells, 2000] Castells, M. (2000). *End of Millennium*. Number 3 in The Information Age. Blackwell, Chichester.

[Chandler, 2008] Chandler, J. A. (2008). Contracting insecurity: Software license terms that undermine cybersecurity. In [Matwyshyn, 2008], pages 159–201.

[Constitution Committee of the House of Lords, 2008] Constitution Committee of the House of Lords (2008). Second Report; Surveillance; Citizens and the State. www.publications.parliament.uk/pa/ld200809/ldselect/ldconst/18/1802.htm.

[Digital Preservation Coalition,] Digital Preservation Coalition. Handbook of preservation management of digital materials. www.dpconline.org/graphics/handbook/.

[Drahos and Braithwaite, 2002] Drahos, P. and Braithwaite, J. (2002). *Information Feudalism*. Earthscan.

[Feller et al., 2003] Feller, J., Fitzgerald, B., Hissam, S., and Lakhani, K., editors (2003). *Taking Stock of the Bazaar: Proceedings of the 3rd Workshop on Open Source Software Engineering*. opensource.ucc.ie/icse2003/.

[Gates III, 1976] Gates III, W. H. (1976). An open letter to hobbyists. www.blinkenlights.com/classiccmp/gateswhine.html.

[Himanen, 2001] Himanen, P. (2001). *The Hacker Ethic and the Spirit of the Information Age*. Vintage, New York, NY. Prologue by Linus Torvalds, epilogue by Manuel Castells.

[Hu and Junnarkar, 1999] Hu, J. and Junnarkar, S. (1999). AOL blocks Microsoft Net messaging. *CNet News*. news.cnet.com/2100-1023-228960.html.

[Hunt, 2000] Hunt, S. (2000). *A general theory of competition: Resources, competences, productivity, economic growth*. Sage, Thousan Oaks, CA.

[Jefferson, 1907] Jefferson, T. (1907). *Writings of Thomas Jefferson*. The Thomas Jefferson Memorial Association.

[Klein, 2000] Klein, N. (2000). *No Logo*. Picador, New York, NY.

[Levy, 2001] Levy, S. (2001). *Hackers: Heroes of the Computer Revolution*. Penguin, London.

[Loren, 2004] Loren, L. P. (2004). Slaying the Leather-Winged Demons in the Night: Reforming Copyright Owner Contracting with Clickwrap Misuse. *Ohio Northern University law Review*, 30(3):495–536.

[Matwyshyn, 2008] Matwyshyn, A., editor (2008). *Harboring Data: Information Security, Law, and the Corporation*. Stanford Law Books, Stanford, CA.

[May, 2003] May, C. (2003). Digital rights management and the breakdown of social norms. *First Monday*, 8(11).

[Raymond, 2001] Raymond, E. S. (2001). *The Cathedral and the Bazaar*. O'Reilly. www.catb.org/ esr/writings/cathedral-bazaar.

[Shaikh and Cornford, 2003] Shaikh, M. and Cornford, T. (2003). Version Management Tools: CVS to BK in the Linux Kernel. In [Feller et al., 2003], pages 127–132. opensource.ucc.ie/icse2003/.

[Shirky, 2008] Shirky, C. (2008). *Here Comes Everybody*. Allen Lane, London.

[Solove, 2007] Solove, D. J. (2007). *The Future of Reputation*. Yale University Press, New Haven, CT.

[Stefik, 1997] Stefik, M. (1997). Shifting the possible: How trusted systems and digital property rights challenge us to rethink digital publishing. *Berkeley Technology Law Journal*, 12(1):137–160.

[Sunstein, 2002] Sunstein, C. R. (2002). *Republic.com*. Princeton University Press, Princeton, NJ. Second edition also available: Republic.com 2.0, 2007.

[Surowiecki, 2004] Surowiecki, J. (2004). *The Wisdom of Crowds*. Doubleday, New York, NY.

[Tapscott and Williams, 2006] Tapscott, D. and Williams, A. D. (2006). *Wikinomics*. Portfolio, New York, NY.

[Williams, 2002] Williams, S. (2002). *Free as in Freedom; Richard Stallman's Crusade for Free Software*. O'Reilly, Sebastapol, CA.