Comments on “Opening the Open Source Debate”

by Julião Duartenn

Director, Security Skill Center
Oblog Software, SA
Espirito Santo Data, SGPS
Portugal

juliao.duartenn@esdata.pt

A commentary on the white paper
“Opening the Open Source Debate”
by Kenneth Brown,
President of the Alexis de Tocqueville Institution

Introduction

Much has been said about the evolution of software development, especially regarding the open source philosophy of code sharing and reuse.

While being contrary to the proprietary nature of closed source development, open source effectively brings software development into the worldwide arena of globalization. No longer a single company holds the source to a product, no longer a customer is left stranded when his software supplier goes bankrupt, no longer a company has to re implement all of the components it need in order to build a software system.

This shifts the focus from “ownership” to “service”, just like the industrial revolution shifted the focus from “land ownership” to “production”.

Open source is an evolution, not a revolution. It won't happen overnight. It started happening more than two decades ago, and it will continue happening, and providing greater value to those who join the movement. Open source is not about philosophy at all. It's about cost, and about benefits. It's about maximizing return on investment while decreasing time-to-market.

It's about people that stop being protectionist and start being productive.

> *Opening the Open Source Debate*
> *A White Paper*
> *June 2002*
> *Kenneth Brown*
> *President*
> *Alexis de Tocqueville Institution*
> *...Like a map to a buried treasure, once you have the code, you can create the software...*

Let us start by this first assumption. “once you have the code, you can create the software”. This shows a serious misconception as to what software actually is. We could rewrite this as “once you have the source code, you have the software”. Source code is, in fact, the ultimate definition of software: An ordered collection of

instructions intended for a computing device, in a form understandable by the computing device, but also, in a form understandable by the issuers of those instructions (read: programmers).

Software must be understood by the computer in order to perform its function. But software must also be understood by the programmers, so that one programmer may understand the function of software written by another, and so that said programmer may introduce changes to the way software should behave, according to his/her specific needs and the computing device he requires the actions to be performed in.

Namely, the desired attributes of software are functionality, correctness, understandability and adaptability.

Since we've cleared that out of the way, let's start with the document itself...

> *Opening the Open Source Debate*
> *By Kenneth Brown*
> *I. In the Beginning....*
>
> *What we know as software is the end result of painstakingly typing thousands, sometimes millions of lines of text in programming language (see Appendix 1). This text is commonly known as ``source code.'' The source code is unrecognizable to most unless you know something about programming, but this text is the ``secret formula'', the underlying schematic that enables each process and function of a software product to work.*

Again, the author seems confused as to the true nature of software. Software is the ordered collection of whatever commands are required to instruct a computing device to perform a function. Source code is not a “secret formula” or a “schematic”. Agree as we must that some sort of analogy may be useful in allowing the general public to understand a technical subject, any serious paper on software cannot incur in oversimplifications if they distort the subject being analysed.
In another note, software does not orbit around “painstakingly typing”. The vision of software must be centered in understanding a problem and devising a solution, and then implementing that solution by way of algorithmic coding. The result is the sequence of commands that will materialize the solution. Source and binary are the same in essence, binary being the result of a deterministic one-way mathematical function (compilation) performed upon the source, allowing it to be converted from human-readable form into machine-readable form. No more, no less.

> *The open source debate is about keeping secrets.*

The author is clearly wrong in this assumption. He does not quote a reference for this assertment, therefore I believe I don't have to quote one, either.

> *Completed (written) software is often locked by its programmer, hiding the underlying code from its user.*

Again, this shows either gross lack of knowledge or an attempt to mystify the reader. First, source code is not “underlying” the binary code any more than a loaf of bread has an underlying sack of grain. One is the product of the other. The baker that makes the bread has no intention of “hiding” the sack of grain from the buyer. The mere concept is ridiculous. The making of the bread is a process of transformation, not of “hiding”. In the same way, compilation is a transformation process, albeit an automatic one. The purpose of compilation is to allow execution on the computer. Not to hide, not to protect. To enable and to materialize.

> *Software can only be modified in its ``unlocked'' state when source code is viewable.*

I am not sure how to read this. Does the author claim that software can only be modified if one has the source? If so, he is clearly wrong.

Viruses are the most eloquent example of this. Viruses change the binaries of software, introducing unintended behavior into a software product, without ever having access to source. In fact, the common targets of viruses are by and large commercial products, whose source has never been discloses to the public. Yet, the software is modified in its binary version.

But viruses are bad examples of software change. There are very few viruses that perform a useful function for the user, of that enhance the functionality of a software product in a desirable way.

In fact, is is very hard to enhance the behavior of a software product without having access to its source. Enhancing requires understanding of what the software is doing and how it is doing it, so one may enhance it by adding the correct feature, in the correct way. This kind of insight does require access to source.

On the other hand, introducing malicious behavior in software can easily be done with the binary alone. It does not require knowing what the software is doing, or how it's done. It just requires finding a point at which the operation of the software can be subverted. And that is several orders of magnitude easier.

In short, software can be modified in any state, source of binary.
To modify it for useful purposes involves collaborating with the software, and usually requires source code.
To modify it for malicious purposes does not need understanding of what software does or how it does it, and can easily be done with binary alone.

> *Software's locked state is also described as its ``executable'' format.*

Again, the notion of the executable version of software being “locked” is wrong. See above comment.

> *Executable software is commonly sold in stores and available commercially.*

Allow me to object. If this sentence had been written the other way around, I would certainly agree that “software commonly sold in stores and available commercially is mostly just the executable version of the program”. But the way that the sentence is written makes it wrong. Executable software is the only kind of software that runs on any kind of computer. Therefore, all software is executable, or has an executable version. This means that all commercial, closed source software is executable, but also that all open source software is also executable (read: has an executable version that allows for it to be ran on a computer), and also that all proprietary, in-house software is also executable.

I may be so bold as to say that, considering different software products, most software in existence may, in fact, be proprietary software developed in-house and not available for sale anywhere.

> *Executable software accompanies binary code also known as machine code. This binary code is readable by the host PC and used to mechanically operate the software.*

This is a very confusing sentence. The author seems to imply that what he calls “executable software” is “accompanied” by “binary code”. This is wrong. “Executable software” IS, generally, “binary code” or “machine code”, with the notable exception of pseudo-code and scripting languages. Furthermore, this “binary code” is not used to “mechanically operate the software”: this binary code IS the software – saying “machine code” is the same as saying “executable software”. The fact that the author makes this kind of confusion preoccupies me as to what train of thought is his reasoning based upon. No person writing about software should be allowed to sustain these kind of childish misconceptions and retain credibility.

> *Developing software inherently creates two versions -- one that is sealed or executable and one that is open source, disclosing the underlying code.*

Again, this is not correct. Saying this is roughly the same as sustaining that “cooking food produces two versions: one that is raw and one that is cooked”. As is obvious, this is wrong. There are no “two versions”. The executable, compiled code is a product of the source code, obtained by direct automated transformation. There is no further intellectual contribution from the author in transforming source into binary. Any 12-year-old can do it.

> *The decision to keep code secret is the prerogative of the programmer. And it is this prerogative that propels us into the open source debate.*

This is the key issue. It's not about having two versions, it's about disclosing the real software product, including source, or disclosing just the compiled product.

Yes, it is the prerogative of the programmer, or of the software house that owns the rights to the finished product. As with any possession, material or intellectual, the owner is free to do with it as he pleases. It's only natural. It's the law of the land.

> *Open source software is not necessarily free software. Programmers sell their software as*

completed products, with or without the code. Likewise, open source and nonopen source products are both sold and available free to users. For example, Apache is an open source web server. Apache accompanies its source code, but it is also used to create commercial applications. Conversely, Adobe Acrobat Reader, a widely used proprietary software for viewing documents is free, however, its owner Adobe has no intention of releasing its source code.

This is correct, although the meaning of “free” should be ascertained. But the author is correct in saying that open source software is not necessarily free, and, may I add, not necessarily gratis, either.

Like a map to a buried treasure, once you have the code, you can create the software.

Again, the same basic mistake. The source IS the software.

Programmers that make a living leveraging the unique value of their software, do whatever it takes to keep their code secret. As expected, most successful programmers and companies do not disclose their code and sell their software without the source code. This is commonly referred to as selling proprietary software. Today, there are several variations of proprietary software produced and sold around the world (See Appendix 2).

“Do whatever it takes” seems inappropriate. So does the word “secret”.
On another topic, open source is not just about distributing readable code, it is also about distributing changeable code and allowing for derivative works, and for changes to be reintroduced into the original codebase.

Programmers have freely exchanged source code for many years. Researchers and scientists promoting their findings would often publish their code. Today, programming enthusiasts have hundreds of clubs, associations and email discussion groups to discuss developments in programming language and new applications. As expected, many of these programmers are able to troubleshoot problems by exchanging their code with one another in these groups. While individuals within each of these groups exchange code for various reasons, completed software often comes with rules for using, copying, selling and distributing the respective code.
Depending on their interests, programmers choose agreements for their products they feel most comfortable with. Today, there are over 30 different licenses (See Appendix 3) with a number of variations (see Appendix 4) that programmers can choose from to distribute their open source products.

Factually correct. I would only like to state that the discussion and exchange of source code is not limited to groups of “enthusiasts” in “clubs” and “associations”. The same discussion and exchange of source code happens between professionals, in private companies, in government institutions, in internationally accredited organizations.

II. GPL Open Source -- The Gift That Keeps Taking

Most open source licenses allow users to freely integrate code into proprietary software without too much difficulty. For example, the MIT license (see Appendix 5) is one of the more lenient licenses and has very few requirements.

It is normal that is should be so. The MIT license's motivations stem from the fact that its creators felt that unlimited distribution of their ideas, algorithms and software products is a positive thing, and did not take it to themselves to limit commercial usage.

However, the license called ``general public license'' (GPL) is the opposite (see Appendix 6). The GPL is one of the most uniquely restrictive product agreements in the technology industry. As expected, the controversy of the GPL is rooted in the language of its license.

This is completely wrong. The GPL is not a product agreement, as is wrongly stated by the author. A product agreement constitutes a license that must be accepted in order to run a product.

The GPL is NOT a product agreement. In fact, the GPL itself states that “The act of running the Program is not

restricted”.

Furthermore, the GPL does not even cover any activities other than copying, distributing and modifying.

From and end-user point of vue, the GPL imposes absolutely no conditions. The user may use the software as he pleases, to do whatever he pleases, on as many machines as he pleases. If you compare this with the current licensing practices of companies such as Microsoft, that require disclosure of personal information before you are allowed to use their product, I cannot begin to see how the author may claim that the GPL is a restrictive license agreement.

> *Its preamble outlining the most major difference from all other licenses begins with the rule that, ``... 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...". The license includes these details and others explaining that if you include GPL source code, the new product also is GPL. Section 6 makes this point very clear stating, "... 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....".*

So far so good, although there are finer points in the GPL that the author “forgets”.

> *The GPL requires that if its source code is used in any type of software product (commercial or non-commercial) for any reason, then the entire new product (also known as the derivative) becomes subject to terms of the GPL open source agreement. For example, if the code for a software application was originally 10 lines, and 5 lines of GPL open source is added to it, then the entire 15 lines becomes GPL open source.*

Let us start by saying that no programmer has ever been required to use any part of a GPL program for any purpose whatsoever. Even interoperability reasons have never required a programmer to use part of a GPL program. The open source community is strongly based on standards and open specifications, and that effectively means that a commercial, closed-source software author can effectively write proprietary code to interact, on a protocol level, with whatever open-source application he wishes to.

Secondly, the example given is absurd.
Aside from the fact that no useful program is likely to be 10 lines long (aside from algorithms, DeCSS in one line of perl coming to mind”), it isn't likely that a 10 line program would require 5 lines of GPL code.

Even if we take it as an example, it does not hold very solidly.

If the GPL contribution to the software is minimal, the programmer should always write the code independently, based on publicly available algorithms and practices.

If the GPL contribution to the program is substantial, this may clearly be the case of a derivative work, in which case the application of the GPL to the whole product is only logical.

Other reasons may exist, though. Interoperability is one. Extensibility is another.
Let's assume that a company wants to produce an advanced spell-checker for an Open Source Office product, such as the one I am using now.

Should the company be allowed to take the source of the Office product, change it by adding the spell-checker functionality, and then redistribute the customized product as an entirely new product, charging for it as if it were wholly the product of the company's work? Clearly not.
So, does this in fact preclude the company from writing the spell-checker? No.
The company can write a standalone application or module, integrating with the Office product by any available protocol interface.

Alternatively, the company can develop a protocol interface for the Office product, keeping it under the GPL, and have the spell-checker interface that. What if the “keepers” of the open source office don't like that? Well, the

company can always fork the distribution, make their own customized version of the Office program under the GPL, and distribute the spell-checker as a closed-source, separate application.

What companies and individuals alike must understand is that the GPL simply imposes some conditions that are not specifically more restrictive than any other license. The GPL merely makes it more difficult for anyone to receive profit from software or parts of software that they did not develop themselves.

> *In effect, if a programmer uses GPL open source in a proprietary product, he agrees that the new product can be changed, modified and distributed as freely as if it were purely open source.*

But, as we have seen, there are many ways to create proprietary products without using GPL code. The GPL merely states that if you want the benefits, you must accept the laws. If you received the code because it was open, you must feed your changes back to open-ness. If you feel that is unfair, don't use it.

> *Thus, the question becomes, ``why would a programmer that sells his product as proprietary software agree to incorporate GPL open source, in effect allowing its source code to be indiscriminately distributed?'' The answer is most don't! This is why there is such heated debate over the GPL.*

This is stating the obvious: many programmers don't use the GPL. Well, if they don't, their code is not GPL anyway. And if there isn't that much GPL code around, then why the debate and why does everyone seem to want to get hold of it and close it?

> *David Wheeler, publisher and expert in Washington on open source and proprietary source comments, ``without licensing the source code in a multilicense format, (referring to other more permissive licenses), it is impossible for GPL to work for a proprietary business model...''*

I have a hard time with words like “impossible”. I don't see this quote cited anywhere in the references, but it must be the same David Wheeler that recently wrote “Why Open Source Software / Free Software (OSS/FS)? Look at the Numbers” on OSOpinion, and a May 2002 essay intitled “Make Your Open Source Software GPL-Compatible. Or Else“.

> *The origin of the GPL license provides a helpful perspective to understand its approach. In 1971, Richard Stallman joined the Artificial Intelligence Lab at the Massachusetts Institute of Technology in Cambridge, Massachusetts. His job was to improve a computer operating system known as the Incompatible TimeSharing System (ITS). Stallman was well known for his negative opinions about proprietary software. In a conflict over building enhancements into an operating system called LMI, Stallman reverse engineered a rival operating system marketed by Symbolics to help LMI build an application to work with the system. Shortly after the LMI incident, Stallman resigned from the MIT faculty to build a free open source Unixlike operating system. He named the project GNU, which stood for ``GNU is Not Unix.''*

In spite of being a fairly accurate narrative, it completely fails to describe the origin on the GPL.

The GPL originated, as is common knowledge, from Stallman's dispute over Emacs with a company called UniPress. Stallman had written the first Emacs in 1975. Later, in 1982, James Gosling wrote a C-based version of Emacs, that would run on Unix systems. Stallman included this C-version source code, with Gosling's permission, in the 1985 version of GNU Emacs.
Later, Gosling sold the rights to Gosling-Emacs to a company called UniPress. UniPress threatened Stallman to stop distributing the Gosling-Emacs code. Stallman was forced to comply, and later had to write his own code to replace the one that had been taken away.

This is the origin of the GPL. A license created so that no person and no company can take public code away from the community, so that no person can write a derivative of anyone else's public work and make it a private money-making venture.

In not including this information, the author effectively turns this entire section into a rant against Stallman, without ever making a point.

> *In 1989, Stallman decided that the open source community should be more organized and founded the Free Software Foundation (FSF). FSF and Stallman evolved the open source discussion into an advocacy group, promoting the idea that all software should be free.*

He also “decided” that he needed more money to support his efforts, and decided upon legally incorporating a non-profit organization.

> *FSF became well known for its position of free software as well as its radical ideas to end patents on inventions. Stallman's group was determined to convince (whoever would listen), that individual ownership of ideas adversely affected technology.*

The author has not understood the point of the FSF case on patents: **Exclusive** ownership of ideas adversely affects technology. *Individual* ownership, the right to be identified as the author of an idea, has been one of the pillars of scientific advancements since before the Greeks.

> *Stallman's Free Software Foundation introduced the General Public License (GPL). The GPL became referred to as ``copyleft'', aptly describing its license as ``all rights reversed''.*

A better way of describing this is given by Stallman himself, when he says that “Copyleft uses copyright law, but flips it over to serve the opposite of its usual purpose: instead of a means of privatizing software, it becomes a means of keeping software free”. It is not about reversal of rights, it's about reversal of purpose.

> *The viral property of the license enabled developers to copy, examine, modify, and redistribute source code freely.*

Wrong. It is not the nature of the license or its means of propagation that enable developers to do any of that. The so much debated “viral” nature of the GPL prevents restriction of rights to developers.

> *Users are free to charge for the cost of distributing GPL products.*

The FSF themselves survive on this very business model.

> *All derivations of GPL code became freely available to anyone that wanted to use it, regardless of the purpose.*

The author previously tried to assert the GPL as being “restrictive”. This does not sound very “restrictive” to me.

> *The copyleft concept moved from FSF to other organizations. However, as other organizations began to form, support for FSF began to erode.*

The author does not sustain this affirmation with any references. His point of view is radically different from mine. I do not see support for the FSF as eroding.

> *The controversial nature of Stallman's position began to turn away his supporters. By the early 90's, open source enthusiasts began to view Stallman as an extremist and fanatic.*

The author, again, does not cite any kind of reference. Stallman is, indeed, a controversial figure. It is possible that some developers (I am still not sure what an open source “enthusiast” is – it's like comparing software professionals to soccer fans – should I describe the author as an “essay writing enthusiast” ?) see Stallman as an extremist and a fanatic. I fail to understand where the author found reference that indicates that the number of people that see him as an extremist is larger than the number of people that support him and his views. Is the number of people that label Stallman as a fanatic larger than the number of people that actively support the FSF in any given year? Show me the numbers, please, before you make such strong statements.

> *The rise in the popularity of Linus Torvalds and the Linux open source operating system began to create new supporters. Ironically, Linux supporters became the biggest proponents of the GPL.*

I fail to see the irony in this. Linux is in fact licensed under the GPL.

> *Although Stallman is a fallen hero in the open source world, most open source products today are*

distributed under the GPL license.

I fail to see the irony in this. Linux is in fact licensed under the GPL.

> *In Wheeler's paper, More Than a Gigabuck: Estimating GNU/Linux's Size, Wheeler documented that in 2000 almost 60% of all open source is licensed under the GPL (see Appendix 7). Wheeler comments, today I would be confident that the number has probably grown to 80%.''*

On June 2, 2002, the same David A. Wheeler reports that according to Freshmeat, 71.5% of the software packages tracked by the site are licensed under the GPL.

> *III. The Myth of a ``Public Software'' Community*
>
> *Widespread support for GPL open source lies in the IT community's frustration with competitive, closed proprietary software.*

“Competitive” has nothing to do with “closed source”. Many open source projects compete with each other, trying to gain market share, exactly like closed source products do. The community is frustrated, above all, with buggy software that they cannot fix, with hardware drivers that the company won't produce and that no one can write because of software being closed, and of software that locks you into a specific platform, a specific data format or otherwise limits your interoperability and the capacity of using the software in your own computing environment, interface with your other software products.

> *But in fact, it is quite common that programmers experiment with open source until they see an opportunity to capitalize on an idea, then embrace proprietary standards. One could joke that open source has been a bridesmaid but never a bride. The story of the web browser is an example of this reality.*

Several examples also contradict this theory, effectively shining light on a different “reality”. See Apache, which implements no closed source proprietary standards, in spite of being the number one web server in the world. See also Mozilla, or OpenOffice.

> *In ``Storming the Gates'', by Nathan Newman, the Internet's ironic history is well documented. President Eisenhower founded the Advanced Research Projects Agency (ARPA). ARPA coordinated research and development on certain military projects and funded a number of college and civilian agency projects. Project MAC at MIT was a governmentfunded project with $3 million per year to create ``timesharing'' computing minicomputer technology (later the home of Richard Stallman). ARPA also funded the Augmentation Research Center (ARC) at the Stanford Research Institute, as well as the National Center for SuperComputing Applications (NCSA). By the early 60's, ARC was experimenting with email, word processing software and hypertext linked documents. ARPA led to an expansion of experts that would spur growth of the Internet.*
> *Project MAC, ARC, and other ARPAfunded institutions formed the collaborative research network that would eventually become the first Internet. In 1969, the first four "nodes" on the network were linked together. In October 1972, ARPANET was publicly demonstrated. ARPA presided over the creation of tools to use the Internet such as the TCP/IP protocol and the File Transfer Protocol (FTP). At this time, the government was encouraging a wide variety of firms to make these kinds of improvements. A number of firms that foresaw the potential of the Internet began pouring millions of dollars into research and development concepts. In the 1990's, one company in particular mirrored the nexus between open source and proprietary software development.*

Jumping from the early 1970's to the 1990's is a bit sudden. The internet had long moved into the private sphere when Mosaic came about.

> *Jim Clark, the founder of computer maker Silicon Graphics and expert on UNIX standards, founded Netscape, and set out to compete directly with the public domain product Mosaic. Netscape became one of the first companies that created proprietary software from the original ARPA funded research. The first Web "browser," Mosaic, was part of a larger federally funded project to create graphics based collaborative tools. Mosaic was first introduced on the UNIX platform in January 1993, and debuted on Macintoshes and PCs eight months later. Copyrighted by the University of Illinois, Mosaic was an open source product and could be downloaded for free by individuals and*

> *companies wishing to use the Internet for internal communications. Through a commercial partner, Spyglass, NCSA began widely licensing Mosaic to computer companies including IBM, DEC, AT&T, and NEC.*
> *Jim Clark hired Marc Andreesen, a member of the Mosaic team, along with a number of other NCSA programmers, to design what the company called, inhouse, "Mozilla, the MosaicKiller." Within six months, Clark's team succeeded in creating a powerful browser with additional features capabilities to move through the Internet at faster speeds than Mosaic.*

This is a clear example of how open source, government funded research can help create innovative software solutions. The fact that Netscape could take the Mosaic codebase and make it better meant that it could concentrate on making it better instead of having to fund a full rewrite. This shows the positive effect of open source. (NCSA wasn't really open source, but Netscape in fact acted as if it were.)

> *Netscape also went beyond the HTML standards embedded in the Mosaic browser and included the ability to display text formatting that didn't exist in the NCSA version.*

Most things can be used either for good and for bad. Mosaic didn't implement all of the published HTML specifications. Netscape did the right thing in implementing the ones that were missing. But, at the same time, Netscape implemented proprietary extensions, which was very wrong, both from a standards and from a user standpoint.

But, more importantly, Netscape never published the source for Netscape. That, in fact, prevented users from knowing how the standards were implemented, what standards were implemented, and what non-standard features were implemented, and how. If the Netscape source code had been available from the start, the user and developer community could have taken steps to make sure that a fully standards-compliant version did exist at all time.

So, Netscape benefited from “open source” (sort of), and did not contribute back to open source. That's exactly what the GPL tries to prevent. That's exactly what the GPL is for.

> *This made Netscape more attractive to users than Mosaic.*

And, since the Netscape code was closed source, the Mosaic team could not go back to the Netscape improvements and re-incorporate them into Mosaic.

> *However, it also meant that web pages designed to work with Netscape would not be readable by other browsers.*

Because Netscape being closed source, its implementation of extraneous proprietary “standards” could not be emulated and replicated throughout the browser industry.

> *Netscape was an aggressive firm. It endeavored to make its web browser the proprietary standard for web access, hoping that it would inevitably become more important than the PC operating system.*

There are many, many aggressive firms. Some focus on the desktop being the most important component in existence. Some focus on the browser. Some focus on having full, integrated control on the overall computing environment, and do all they can to make this happen. The open source movement actually doesn't care, as long as the protocols are public, the document structures are public, and, where appropriate, the source is free.

> *Netscape began distributing its browser free to users. This strategy all but eliminated interest in Mosaic and NCSA led standards. Because Netscape was also able to do this without paying licensing fees to NCSA, it was able to undercut other commercial browser companies that had to meet NCSA license fee requirements.*

This is the most funny piece of spin I've heard in years.
First of all, Netscape didn't start distributing its browser for free right from the start. By the time Netscape started distributing the browser for free, Microsoft had already included Internet Explorer, for free, in Windows.

And Mosaic... Mosaic was long dead by then.

As for the subject of licensing fees, Netscape was wrongly using NCSA concepts, and reportedly source, yes. That disputed was later settled out of court.

But this illudes the real message: “it was able to undercut other companies that had to meet license fee requirements”. If access to the NCSA codebase were free, Netscape could not have leveraged its illegal position in order to reportedly bully other companies out of the market.

The author is, I believe, trying to prove a point against open source. On the contrary, the point he makes is totally in support of open source.

If the source is open, you cannot extend it with proprietary additions, since they become publicly available.

If the source is open, you cannot harm your competition by dumping prices below cost, and charging everybody else a license fee.

> *Not only did Netscape crush competition with its free browser model, but it also infuriated members of the open source community by aggressively introducing proprietary standards to the public Internet, something they felt no one should own.*

This is something completely different, and has little to do with competition, of prices, or NCSA. Netscape did introduce proprietary extensions. Had Netscape been an open-source product, those extensions would have been easily incorporated into other products, and the impact would have been much smaller.

The problem is not when someone innovates. The problem is when a company uses it's de facto monopoly and leverages it into breaking the standards that insure interoperability and a level playground for all.

> *Conveniently, Netscape turned its enemies to Microsoft and their new browser, Internet Explorer.*

No, sir. Wrong again. It was the other way around. Microsoft was the one that felt threatened and started dumping its browser, leading to the Netscape response.

> *Newman's paper and the Netscape story however prove that public private partnerships are valuable to technology efforts.*

Yes they are, if handled in a responsible manner, and if the output of such efforts is available to all. If you allow a public effort to become a benefit for a single company and not for the community, the amount of good you are doing becomes very limited.

> *While government and research institutions are more efficient in longterm projects, the private sector is more efficient with creating solutions quickly.*

This is a well known fact. The quality and durability of these solutions, however, is open to debate. Do bear in mind that open source development is quite compatible with the private sector, in fact, most open source companies in existence are privately held.

Not also they create solutions quickly, but the output of their work is validated by others, enhanced by others, tailored by others to specific needs and environments. Closed source is quick with solutions, but always seems to cater for the mainstream market and forget the vertical market applications.

> *Conversely, it would be impossible to expect the efficiencies of the private sector to appear in the model of the public sector, just as it would be inane to expect the government to operate like a business.*

This remark seems out of place in what should be, above all, a technical report. The government should in fact learn from the private sector and create, whenever possible, efficiencies using the lessons learned by businesses for the benefit of all citizens.

> *However, it does make sense for each to respect the interests of the other and work together.*

Yes it does.

But don't use this simple assumption in order to justify what you couldn't do for the last 16 paragraphs.

Ok, on to the interesting part.

> *IV. The Government and the GPL*
>
> *The development of the Internet is one example of thousands of successful partnerships between various federal agencies, academia and private enterprise.*

Most of them involve the government funding research and development activities, retaining the right to do whatever it wants with the output of that work.

> *However, the use of the GPL has the potential to radically alter a very successful model for partnership, particularly when most large commercial entities do not readily embrace the GPL.*

This has no grounds. The partnership model in question relies on the government issuing research and development contracts to commercial entities and academia. Would these entities turn the contracts down if the resulting work were to be covered by the GPL?

> *Federal agencies face a number of serious challenges with respect to using GPL source code. Without exception, the license clearly applies to any entity, including the government.*

Federal agencies also face a number of serious challenges with respect to contracting to commercial entities. But these challenges have been met, and all issues have been addressed in the past. If the government wants to use the GPL, the challenges will, again, be met and dealt with.

> *David F. Skoll, President or Roaring Penguin Software, Inc. insists the same, ``...Yes, the government should use and distribute GPL'd software. If the government uses GPL'd software and distributes binaries, it's obliged to distribute the source code.''*
>
> *Unlike proprietary software, opensource is intended to be studied and improved by other programmers, thus it is ideal for government research projects. However, most open source is the end result of hundreds (perhaps thousands) of programmer's ideas, changes and modifications. Free distribution and public exchange of source code enables its innovators to try unlimited ideas for applications.*
>
> *Open source could be described as a virtual laboratory for experimentation.*

With one particular difference from just a “laboratory for experimentation”: it produces viable, usable products, which are, in their large majority, appropriate and cost-effective.

> *Needless to say, the government could not depend on patches for software glitches to ``wander in'' from the public.*

The same applies to closed source software. The government must depend on patches from vendors, some times having to wait more time than it would take to develop a patch independently. If the government only accepts closed source software from “qualified suppliers”, why should it accept open source software, or open source patches, from anyone? In fact, software will need to be certified or provided by a certified party. Many commercial companies can be certified for this purpose. If anyone needs a business model for open source software, here is one.

> *Likewise, the government could only use open source code that it could independently service in case of an emergency.*

This has no grounds. Does the government today use only closed source software that it can independently maintain in case of an emergency? No. It must rely on vendors. And vendors are not very reliable, as we have seen from the latest wave of bankruptcies, companies being sold and employees being laid off.

> *Agencies without extensive staff to maintain its internal operations cannot afford to use hapless and*

untested software without accountability, warranties or liability.

This has no grounds.
First, how many people have tested a commercial product when it ships?
What kind of tests were performed? What measures were taken to correct the issues found?

Open source software undergoes as many tests as commercial software, if not more. Furthermore, test results can be handled in a much more transparent and responsible way through the open source method of development.

Second, liability. All closed source software license agreements explicitly exclude any liability, including the license to sue. Warranty exemptions are the rule. The software company is never liable or accountable for anything the software does or doesn't do.

Another consideration for the U.S. government is that all source code developed under the GPL could have mirrored availability to the public.

As far as non deliberate disclosure (code stealing or accidental disclosure) is concerned, there is no difference from closed source software to open source software. If there is a difference, it is in favor of open source. Open source is, by definition, designed to be world-readable, so no secrets are embedded in it. Closed source, on the other hand, comes with the temptation to embed secret data, and to achieve any degree of security by means of clever tricks, and not through proven algorithms.
As far as being forced to expose sensitive code, provisions exist to separate between GPL code and specific proprietary modules. Well defined interfaces, isolating sensitive modules, may be used in order to create a degree of independence between GPL code and proprietary code, while maintaining integrity of the GPL code. For example, a program that controls the cockpit of a fighter plane may include a generic, robust implementation of everything except the code that activates electronic counter-measures, and the code that controls target-acquisition, and the code that implements pilot decision support. Entry points for all of those could be provided for in the main, GPL, program, but the GPL version of the program may include only dummy versions of these components. The components themselves may be provided in custom, proprietary form.

This poses unlimited security issues.

The previous comment shows that it may not pose any additional risk when compared to the current situation.

Wheeler comments, ``There are many programs developed by the government which are THEMSELVES classified, and many and probably most of the various programs most important to national security are in this category (e.g., weapons systems). In that case, neither binaries nor source code of those particular applications are released to anyone else; besides being illegal, releasing the binary executables would give away far too much information."

As mentioned, the key lies in isolating the sensitive and classified components. This may, in itself, prove beneficial insofar as, provided with a clear interface between the components and the core program (required to separate GPL from closed source), the components may be commoditised, reused, contracted to different suppliers, and so on.

Rossz VamosWentworth a programming expert, disagrees about the relevance of releasing code commenting, ``If the government uses GPL software, the government is to release their version(s). If the software is related to security, it really doesn't matter if the code is available or not. Security holes are eventually found, with or without open source code. If the security software is well done, having the source code will not make it easier to ``crack''.

The issue with security in government software is wider than the issue with security in the corporate world. While software is seldom used in the corporate world to implement proprietary business practices or methods (software is usually generic, the relevant business parameters lies often in data), in the government scenario, especially with military systems, this may well be the case.
Take the guidance system for a missile. The system is programmed to respond to certain sensors, according to certain thresholds. If you don't know what the thresholds are, you have more difficulty in building a target which stays under those thresholds. You have to go by trial and error. If the source code (or even the binary, reverse engineering is possible if you want it bad enough) were to be revealed, it would be orders of magnitude easier to

build such a target.

But the guidance system for a missile is unlikely to be based on GPL code. Furthermore, it is unlikely that is would ever be distributed to third parties. Which would not trigger the GPL into action. Even if it were based on GPL code, the mere fact that it is not distributed to anyone would make it as safe as proprietary code.

> *Defense agencies have obvious concerns regarding sharing publicly its programming architecture for critical military systems. But other federal agencies have similar cause for concern? Jason Rexilius, an exUSAF intelligence analyst currently an applications architect contrasts and compares the government's decision with the private sector commenting, ``...One might be that the algorithms used in the code would be classified. This is common in the private sector and is a valid reason for keeping certain source closed. As an example an algorithm developed by an auto manufacturer used for testing shocks may be kept as a corporate secret rather than patented because patents are in public domain and it would be difficult to prove that a competitor was using it.''*

Again, the answer is code isolation. Build the entire system as GPL, but create a clear interface where the proprietary closed source algorithm will plug-in. Then create a dummy plug-in, to show others how to do it and to prove it can be done. Then, distribute if you have to, keep it if you don't, but the algorithm remains closed source.

> *The federal government's information systems requirements intersect countless sensitive operations. The limitless potential for holes and back doors in an open source product would require unyielding scrutiny by staff that decided to use it.*

Open source software does not have any more potential for holes and back doors than closed source software. Furthermore, with open source software the source may be inspected if there is reason to believe in the existence of back doors.

As for holes, the majority of vulnerabilities found in any mailing list is for closed source products, not for open source ones. And the mean time for a fix to appear is far, far less in open source products than in closed source ones.

> *For example, if the Federal Aviation Agency were to develop an application (derived from open source) which controlled 747 flight patterns, a number of issues easily become national security questions such as: Would it be prudent for the FAA to use software that thousands of unknown programmers have intimate knowledge of for something this critical? Could the FAA take the chance that these unknown programmers have not shared the source code accidentally with the wrong parties? Would the FAA's decision to use software in the public domain invite computer ``hackers'' more readily than proprietary products?*

No. The FAA's decision to use Windows will invite “hackers” more readily than any other product, proprietary or not. Believe me. It will.

> *Outlining a noteworthy scenario, Rexilius comments, ``...You have a closed network in your building with guards, cameras, and ID badges to prevent physical access. You also have proper security INSIDE your network utilizing strong passwords, encryption, and sound wellbuilt applications. Now if you open the blueprints for every aspect of it to the world your adversary can reconstruct a test lab in which he can create tools he may need. He then manages to gain physical access to your network by posing as a janitor and has a 5 minute window in which to try and crack your system. If you have a wellbuilt system this will not matter however if you have security problem and the attacker had prebuilt his tools this would be the window he needed...''.*

This scenario only applies if you are running proprietary software and you are the only one in the world to use it.

Using open source does not mean that you must disclose your network topology. Nor that you must reveal what algorithms you use for encryption. Believing that using GPL software forces you to disclose all and every bit of information about your systems is ridiculous and wrong.

And if your network and systems can be compromised by the mere fact that anyone can build a lab like yours, and the solutions they come up with are implementable within a 5 minute window with physical access, then it doesn't matter what software you are using. You're not handling security right. Hire a professional to do the job.

Going back to using software that is unique, meaning that you do not use commercial off-the-shelf software but use proprietary highly customized versions, this is very very seldom the case.
Most sites use COTS operating systems, databases, applications. These can be easily used to replicate the said test lab, as easily or more easily than any open source software. If you use open source, you can modify settings, parameters, even rewrite functions. With closed source, you're stuck with the standard distribution. Which is the higher risk, in this scenario?

Another issue: if you do use closed, proprietary software built just for you, doesn't the reliability issue come to mind? Just a while back, commercial software was said to be very good because it was tested. Testing means having been used, extensively, in many conditions. If you're going with a proprietary one-shot solution, the odds are you haven't tested that much.

> *Needless to say, the aviation agency is not the only department of the federal government that would have these security concerns. The Securities and Exchange Commission, the Internal Revenue Service and the Federal Reserve all monitor, collect and store sensitive information.*

Information! The word here is information! It has nothing to do with the software used! You are not forced to disclose a bit more of information for using PostgreSQL than you would be by using Oracle or DB2.

> *In the end, public domain software has its advantages to develop prototypes, but it also creates exponential security issues for agencies that must avoid a breach of its IT systems at all costs, especially when an IT system engineered with GPL software could easily be targeted.*

This conclusion is flawed, since it is based in so many wrong assumptions, as we have proved. GPL software brings advantages in robustness stemming from widespread testing, and in response time to vulnerabilities. Furthermore, GPL software is usually widely understood by the community at large, and tends to implement tries and true algorithms and procedures.

This conclusion also contradicts what has been said about the speed of the development life cycle, when private companies were being touted as being quicker and more efficient. If anything, I would draw the conclusion the other way around: closed source software is good for prototyping, for demo systems, but should never be taken into the real production systems. Production systems require hardening, testing and transparency that is only possible with an open source development model, or by significant investment in testing and modelling.

> *Experts differ on whether the primary focus for security should source code or binary code. Andrew Sibre, a programmer with over twenty years of experiences insists, ``Having a license for binaries only gives you a ``black box'': you don't know what it's doing, or how, unless you want to go insane trying to reverseengineer it with a debugger (illegal under the term of most licenses... Having the source lets you see what it's doing, how it does it, and permits you to modify it to meet your particular requirements (including security related ones).'' To this extent, government officials should be concerned that threat may not just be an adversary cracking their system, but inadvertently educating adversaries about their security systems. Sibre continues, ``...Depending on code without the source is quite similar to depending on a complex mechanical or electronic system without the benefit of shop and parts manuals...''.*

The argument is the same as we have mentioned: closed source is harder to extend and improve upon, while still being as easy to break. Take the “complex mechanical or electronic system” example. It is very hard to try and improve on a television set without having the schematics. It is, however, very very easy to break it. All you need is a hammer, no manuals.

> *IT systems may depend on the protection of both the source code and the binary code.*
> *Wheeler continues, "If source code is the "blueprint", then the executable (binary) is the "building". Why would an attacker need a blueprint, when they can get an exact copy of the building to explore and search for vulnerabilities? And in the world of software, an attacker can automatically generate the blueprint from the building. Protecting the blueprint but not the building makes little sense."*

As for protecting the binary code as well as the source, I totally agree. If you must absolutely depend on secrecy of the algorithm or the procedure, which only happens in limited situations (see missile-guidance), you cannot afford to disclose either the source or the binary. But that also excludes commercial software, since the binary of that gets distributed. The only solution is in-house development, even if backed by contract work. The GPL applies if you try to distribute software, not if you use it for your own. It doesn't matter if the system is based on

commercial software or on open source software.

> *Another security concern is that the primary distribution channel for GPL open source is the Internet.*

The primary, not by all means the only source. And this is not a security concern. Many ways exist to ascertain the correctness of the software.

> *As opposed to proprietary vendors, open source is freely downloaded.*

So is many commercial software, most patches, innumerous drivers. Don't those constitute security problems?

> *However, software in the public domain could contain a critical problem, a backdoor or worse, a dangerous virus.*

So can commercial software, and in fact already has in the past. The risk is nor higher for open source software. In fact, it may be lower, since open source programs can be inspected and compiled by the user, where closed source programs can not.

> *Downloaded software would require resources to perpetually screen its source code.*

The required screening for code is the same as the required screening for binaries. You must trust the source if came from, be it IBM, Oracle, the Apache Software Foundation or Bruce Schneier. The code must be signed, either by a certificate or merely through a software hash. You must be certain that the immediate source (ftp site, download.com) is reliable, and must always verify the software hash fingerprint versus the one posted on the original development team site.

> *Carey Lening, Student at Pierce Law Center in Concord, NH offers a simple analysis commenting, ``This all boils down to a matter of trust. If you trust the source boards and volunteers, the patches can be a great asset. Provided people don't have bad ends, the immediate, large aspect of a community where folks can check out the work of others is good. The downsie only comes when the community either a) doesn't check their work or b) doesn't have very high morals..."*

The same issues apply to commercial software.
Commercial software has been released with bugs, holes, security issues, falling into a).
Commercial software has also been released containing trojans, spyware, tracking software. Falling into b).

> *The government's productive alliance with private enterprise is also relevant particularly when its decision to use GPL source code would inherently turn away many of its traditional partners.*

As we have said before, no it would not. Most if not all of the government contractors would be more than willing to work on a GPL codebase in order to produce the required contract.

> *Security, as well as other impracticalities make GPL open source very unattractive to companies concerned about intellectual property rights.*

All the security issues raise have already been dismissed, and intellectual property should never be an issue when discussing government contracts. If the product being bought is a COTS product, there is no added value or loss in not buying it and using an open source product instead. The company's IP will not be reduced in any manner. If, on the contrary, the contract is for specific development, the IP for the output should always remain with the government that has funded the activity, and therefore there is no loss for the company in disclosing the source of the software.

> *In effect, the government's use of GPL source code could inevitably shut out the intellectual property based sector.*

There are no grounds for this statement, It is gratuitous, and probably wrong. On the contrary, more intellectual property is being generated by the open source community than by closed source development. And the positive effect of that intellectual property on the economy has been far greater than the positive effect of any closed source implementations. Most of the things that provide the most benefits are public.

This has a number of ramifications. Immediately, it would limit the number of qualified vendors to choose from to deliver products.

The number of vendors dealing in open source software is quite large, and it has a far greater potential for growth if the government embraces open source than it is likely for closed source businesses to close. The number of qualified vendors to choose from seems to diminish more from the mergers (HP+Compaq) and the acquisitions (see Microsoft's history of acquisitions) than from any adverse open source impact.

The GPL's wording also prevents the equal use of software by members of the IP community and the GPL open source community.

This requires clarification. As stated, it is void and wrong.
There is no technical distinction between a “member of the IP community” and a “member of the open source community”. Many people are both. In fact, most people are both.

A worse consideration is that use of GPL could inadvertently create legal problems. IP community members could argue that the government's choice of open source is restrictive and excludes taxpaying firms from taxpayerfunded projects.

The government's choice of open source is as restrictive than the government's choice of closed source. Any company can work on open source projects. Claiming an option for open source as restrictive is like claiming that the government is restrictive for not buying umbrellas in summer, excluding taxpaying umbrella firms from government money.

Any firm can work in open source. It's a matter of choosing to.

Adverse impact would include a discontinued flow of technology transfer from governmentfunded research to the technology sector. Without value, it becomes highly likely that government funding for research would slow as well.

Wrong again. The government would likely be continuing to contract the work that is being done in research. What the government would probably not be doing so much would be paying license fees for products that will not be specifically addressed to its needs. The flow of money from the government to the private sector would obviously continue. But now, instead of paying for license fees the government would be paying for real development, for real expertise in implementing systems, and for more research.

V. ``Intellectual Property Left''

U.S. intellectual property (IP) statutes have been a beacon for inventors around the world. The U.S. model for motivating, compensating and protecting innovators has been successful for almost 200 years. GPL source code directly competes with the intellectual property restrictions, thus it is vital to analyze its impact.

The true value of IP as a stimulus for innovation in today's world remains to be assessed.
At the same time, public sharing of knowledge in the academic community has been a beacon for inventors around the world, too. So the benefits from the adoption of one above the other remain to be quantified.

Our belief is that both models, combined, provide a better framework for the advancement of science and technology. Thus, the existence of GPL code further contributes to this positive diversity, allowing different business models to thrive and maximizing the return on the world's intellectual capital.

The impact is therefore positive, as the GPL does not “compete” with any other models, it rather complements the existing frameworks for software creation.

There are two groups of programmers that contribute to the open source community. The first group consists of professionally hired programmers by day, who freely contribute code. The second group consists of original equipment manufacturers (OEMs) that are hiring open source programmers for their products.

> *However, open source principally perpetuates itself because there is an avid pool of experts and enthusiasts willing to spend their spare time to provide ``fixes'' and modifications to opensource software. This volunteer system works well as an academic model, but not as a business one. It would be tenuous at best, to expect a volunteer model to compete with a forprofit model.*

This is a contradiction in itself. Many companies have vested interests in open source products, making a profit from services and skills, as well as complementary commercial software and hardware. The open source model has long stopped being a complete volunteer effort.

There are hundreds of companies whose entire business model revolves around open source products and associated services. These companies have effectively implemented a for-profit model around open source. The “tenuous” expectation of the author is not only quite solid, it has already happened.

> *The underlying premise of capitalism is that compensation is the best universal tool for motivating individuals to succeed. Fulltime programming teams produce innovations for pay and turn to IP protection to market their products.*

“Universal” tools are not the only tools available. In fact, specialized tools often perform better than universal ones.

The software community is a specific one, where specific rules apply, together with the general rules. This means that direct monetary compensation is not the only means by which programmers get rewarded.

IT professionals can often earn money outside their primary activity by performing consulting work, specific contract work or teaching. The money earned in these activities is proportional to the professional's expertise and reputation. Open source projects go a long way into providing a professional with both of these assets. His open source activity, albeit volunteer, becomes a pillar of his paid work.

IP protection may be good for companies, but it seldom brings as many benefits to the individual as sheer peer recognition, expertise and reputation would.

The times have changed in the IT world. We are slowly moving from mass-market corporate capitalism to individual capitalism, to citizen's capitalism. The values are slowly shifting. Corporations, of course, may not like it, for they will need to adapt. Individuals, on the other hand, will embrace their rise to power.

> *However, without an incentive to create commercial software, filings for copyrights and patents would immediately decline. Thus, it can be expected that innovation would be adversely impacted if the financial incentives for innovating were affected.*

Filings for copyright and patents are not a measure of the economy's well being. The patent process in itself has been much discredited, mostly because of the Patent Office's inability to discern between patents that should be granted and others who should not. And even the copyright mechanism has been repeatedly subverted by political manouevres so many times that its intrinsic value is becoming questionable.

Innovation in the academic community has never seen any kind of decline. Academy is still one of the largest innovators, on a global scale. Still, there is no financial incentive for innovating in academia. This, alone, proves a fatal flaw in the author's reasoning. There are other incentives besides financial ones.

> *As mentioned earlier, open source code is not guaranteed nor does it come with a warranty.*

No code, commercial or open source, is guaranteed. Commercial companies disclaim all warranty and liability for software, always.

There is in fact no difference between the kind of warranty or guarantees provided with open source software and with commercial software. This argument, therefore, has no grounds.

> *Open source products are often distributed without manuals, instructions or technical information.*

Amusingly, this is not a problem about open source software at all. Most commercial products are distributed with no manuals, instructions or technical information. Operating systems are a good example. In any case, open source products sometimes provide as much or more technical information and documentation as their

commercial counterparts. Linux is a good example, Apache is another. The issue may well be the quality of the documentation, or its orientation for a technical audience rather than a less knowledgeable user base. Nevertheless, great progress has been made, and very good documentation exists in some open source products, especially the more mainstream.

> *While a commercial developer is obligated to produce manuals, diagrams and information detailing the functionality of their products, open source programmers are not.*

Quite on the contrary, most commercial products do not describe their form of operation at all, providing mere user guides outlining basic functionality. Open source products, due to the distributed nature of their development process, sometimes provide much more detailed and richer information about their internals and schematics as commercial products.

Additionally, if custom development work is considered, the issue is simply irrelevant. The documentation produced will be the documentation contracted, regardless of the source availability model or the codebase.

> *In addition, open source developers cannot be expected to create software manuals with the vigor of private firms that are obligated to produce them. Producing technical specifications (in soft or hard copy format) is timeintensive and expensive.*

“Obligated” is the wrong word. Commercial products only have manuals if competing products have manuals, in most cases there is no contractual requirement whatsoever to produce or deliver documentation for a finished product, unless of course the issue is custom development work. The depth and accuracy level of documentation in commercial products has decreased with the advent of near monopolist positions in the market. Documentation is more and more targeted towards a sub-par user base, outlining basic functions in extreme detail, but more and more lacking in deep technical documentation about the core functionalities of products.

Commercial products carry more and more “hidden features”, “special settings” and convoluted ways of specifying behavior, none of which are documented, This is seldom the case with open source software.

Yes, producing technical documentation is time consuming and expensive. But so is development. And if the open source model has successfully managed to produce vast bodies of software using the latter, its ability to provide rich bodies of documentation is effectively asserted as well.

> *But this is not just a customer service issue. GPL open source or ``copyleft'' reverses the intellectual property model. For example, in the real world, it would be absurd for users of a technical manual to have permission to modify it in any way and republish it as theirs. More importantly, when a manuscript is bought, because of the enforceability of copyright law, we know that it has not been altered in any way, thus the copyright ensures a buyer (or user) of the manual that he has a genuine copy of the real McCoy.*

In the real world, many users annotate their own commercial manuals with shortcuts, work-arounds and examples. In the real world, many of these users build web sites detailing procedures not found on the manuals and providing general help and frequently asked questions.

In an open source world, these users contribute back to the documentation pool, and their changes, additions and suggestions are incorporated into the standard documentation and redistributed, benefiting all of the user population.

The matter of ownership of documentation is a non issue, understood only because of a narrow view anchored in out-of-date concepts of feudal property. Copyleft does not go “against” copyright law. Copyleft uses copyright law to attain different objectives. Copyleft fully asserts the rights of authorship and guarantees the integrity of content. Any changed copy of a piece of software or documentation must, as per the GPL, clearly state all alterations made to the previous version. A user interested in the “official” version of a manual will obtain it from the same entity he got the software from, usually the distribution's maintainer. Together with the manual, a digital hash will be provided so that the user can ascertain that no spurious changes have been introduced to the original text.

Copyleft is not about “taking something and republishing it as theirs”. In fact, this is forbidden by both copyleft letter and spirit. Copyleft is about extending, about correcting, about improving. If the user would rather go for “John Wayne's Manual for Apache” instead of the Apache Software Foundation version, he has a choice. But this is exactly what happens in the commercial world. If a user prefers “Oracle for Bozos” instead of the mandated

Oracle official manual, this is his choice and his option.

> *Innumerable questions surround the distribution of technical information in the copyleft environment, particularly because the Free Software Foundation has a copyleft license for its documentation as well. Issues include: Who should have the right to alter software manuals? Who is the final editor or is there one? How should changes be regulated? Are manuals copyright protected documents? What is the process for making changes? What body regulates these changes? How can organizations guarantee that information in manuals is always accurate?*

All these issues have solutions and answers, which I will not detail here but upon which I am available for further comment. None of them pose any kind of conflict or issue, and all of them have simple solutions that benefit both the user and the author. Just a comment on the “accurate” part: Do commercial entities guarantee that the information in manuals is accurate? Is this always the case? The answers are “no” and “no”.

> *Today, software impacts a firm's financial health in an intimate fashion. It becomes unrealistic for a firm to depend too much on the ``trust'' of an anonymous community that does not have anything at stake financially to keep important technical documents current.*

The issue is the same for documentation as it is for software itself. The open source community does have something at stake, albeit not strictly for financial reasons. Technical documents need be changed whenever the software changes. User manuals for commercial products often fail to comply with that, too.

> *The reliance on reverse engineering is probably one of the biggest conflicts between the IP and the GPL open source community. To keep GPL products relevant and up to date, GPL enthusiasts must perpetually reverse engineer intellectual property.*

Reverse engineering is a widely accepted practice, namely for interoperability purposes. The only reason reverse engineering is necessary is because of vendors that do not disclose required information about their products in order to make them work with other products and in diverse environments. This is like selling a car that only runs on special fuel and refusing to say what goes in the fuel, so that you are the only seller in town.
Non disclosure of interoperability information is a serious offense and a serious threat to the free market and to the very foundations of capitalism. Without disclosure, and without reverse engineering, there would be no standard electrical sockets. There would be no auto parts market. There would be no construction materials industry as we know it, because nuts from brand X would not fit bolts from brand Z.

If the open source community must reverse engineer formats and interfaces in order to achieve interoperability, it's the commercial industry's fault. Reverse engineering for interoperability furthers the advancement of science and the technical panorama. Not disclosing interface information tries to lock the user into a vendor monopoly, and is harmful for consumer choice.

Open source is, in fact, about choice. It's about being able to choose the best for yourself and for your family, and for your company. It's about not being locked into a specific vendor or group of people just because you bought something from them in the past.

Every product sold should come with interoperability information. Corporate greed or blindness sometimes prevents this. That's why the community reverse engineers. Because we have to, not because we want to.

> *For example, many hardware manufacturers (also known as original equipment manufacturers or OEMs) keep hardware specifications secret. This makes it difficult to share source code about the product in the public domain. For example, hardware's design determines its printing specifications. An OEM's new printer accompanies software known as a ``driver''. Once loaded, the driver enables the PC to communicate with the new printer and print documents. Open source enthusiasts reverse engineer the driver and make the code available in the public domain. Once a driver is reverse engineered, programmers can now write software to enable their software or devices to communicate openly with the OEM's printer.*

This is a paramount example of what I stated in the previous paragraph. The driver implements the interface between any software and a particular piece of hardware. As such, it enables me to use the hardware I choose with the software I already own.

Without a driver, I can not use a particular piece of hardware. It is only natural that the hardware vendor would

like me to be able to use his hardware with whatever software platform I am using. This would maximize the potential user base, and would not lock users of of their market space. But this is often not the case. Vendors sometimes only provide drivers for one or two computing platforms or operating systems. This prevents users of other operating systems from using that hardware. This limits choice. This also limits revenues for the hardware vendor.

It doesn't make sense for me not to be able to buy a new printer because it is does not support my operating system. It doesn't make sense that I have to choose an inferior printer product because of my needs in terms of operating environment. And it does not make sense that I have to change my operating environment in order to be able to work with my printer. People don't renovate their houses in order to fit the dish washer.

If the hardware vendor would publish the interface specifications for his hardware, it would be much easier to write a driver for a different operating system. If he doesn't, the only way to ascertain those interface specifications is by reverse engineering.

Some vendors happily provide interface specifications to whomever asks them for them, including open source developers. Drivers for their products appear quickly on the market, and sometimes the vendor itself distributes the open source driver with the product. Everybody profits.

> *Reverse engineering has a number of implications. It harbors very close to IP infringement because and has staggering economic implications. If software is freely reengineered, it will inevitably impact the value of software on the market. If the price of software is adversely impacted, salaries and inevitably employment of software programmers would be negatively affected as well.*

This is wrong. Reverse engineering is nowhere close to any sort of IP infringement. The right to reverse engineer for interoperability purposes is ascertained both by law, especially in Europe, and by common practice of the science and industry communities. And never has further interoperability diminished the value of any product, or impacted its value on the market. The only thing interoperability grants is choice. And choice is the basis of the free market and of the capitalist system. The free market will never impact products with real value, their price must always be fixed by the laws of that market, and not by monopolistic abuses to leverage inferior products. The market has never allowed that, and it's not going to start now.

> *Commenting on copyright law and reverse engineering, Noel Humphries, senior counsel at Akin Gump's licensing practice in New York explains, ``The source code that the software writer writes (by making it up out of his head) is covered by copyright by virtue of its being typed into the computer. Regardless of what happens to it later, that bit of code is protected by copyright law from unauthorized copying. Think of a book. The particular sequence of the book's words are protected, whether the book ever gets published or not. The point of a license is to set the terms by which someone can possess a copy of the software, while the copyright holder retains "ownership." In that circumstance, ownership is limited by licenses granted to others. "Ownership" may be attributed to someone, but someone else has legal rights to possess a copy and do something with it. For example, a company may come into possession of software that allows the user to see the code. However, the legal right to see the code does not necessarily create legal authority to modify the code or create a wrapper around it. You would have to look at the terms on which the firm came into possession of the code to determine what the firm was legally empowered to do with that code." "Copyright law covers the source code. In fact, the humanreadable code is exactly what copyright law covers, although code compiled in machinereadable form may be a derivative work of the humanreadable code. Copyright is supposed to encourage publication. Copyright law forbids unauthorized copies. Viewing source code is not typically a violation. Taking inspiration from it is not typically a violation. Copying without authority is, Humphries added."*

The Noel Humphries comment is truly about copyright, but it does not say a word about reverse engineering. As far as copyright is concerned, all that he states is true. Yet the GPL is in itself founded upon copyright law, insofar as the GPL does not change, or oppose copyright law, it merely uses it for a different objective. This paragraph, in fact, shows how well the GPL actually achieves its goal.

> *As backlash against reverse engineering, proprietary software companies circumvent open source and GPL licenses, further building distrust amongst both parties.*

While reverse engineering has solid legal and practical background, circumventing the letter or the spirit of a license agreement is a felony, and should be treated as such, since there is no legal or moral standing upon

which the circumventions may be founded.

The author acknowledges that commercial software companies illegally circumvent the licenses, and he further states that this happens as “backlash”. Whether it does or not remains to be ascertained, but in any case, “backlash”, like “revenge”, are two of the lowest emotions and motives known to society, and can never be used as acceptable reason for any activity. Let's hope that the industry's vision is far more ethical and morally sound than the ADTI.

> *There are a number of approaches that are commonly used to bypass GPL license restrictions. Companies choose to work directly with the author of a product to produce a second product (similar to the first) under a different license. Another strategy is to reverse engineer a GPL product in a different programming language, that way, there would exist two completely different sets of source code for the same product. Another strategy is a technique called dynamic linking. This is best described as using GPL code to create a new product, but the derived (new) product's architecture is such that it is only linked to the GPL software. Files in the two software applications interact and enable the two products to work together. The derived work and the original GPL work can exist under two separate licenses. However, the new software will only work in combination with the previous product, not separately.*

These are the canonical forms of circumventing the GPL. The author seems to forget another one, “outright theft”, which is reportedly employed by some companies. Since their product is closed source, there is seldom a way to prove that this does, indeed, happen.

> *On a lighter note, while many open source enthusiasts are proponents for copyleft, they insist on trademark protection for their ideas. The Free Software Foundation, the Open Source Initiative and a number of other organized GPL enthusiasts protect their ``marks'' by posting notices in publications and on their websites that their trademarks are protected. For example, the notice on the OSI website reads, ``... To identify your software distribution as OSI Certified, you must attach one of the following two notices, unmodified, to the software, as described below. ``This software is OSI Certified Open Source Software...OSI Certified is a certification mark of the Open Source Initiative...'' The same is true for a number of prominent open source firms including VA Linux. While each of these firms would insist that they are not against copyright protection, invoking the protections argues that they are against people copying their marketing documents and symbols.*

The author must understand that copyleft is not the opposite of copyright. We knows it's difficult, but let's try again. Source is like a house, which has a door. This door has an old-fashioned lock. If you turn the key on the lock with the door closed, the door will never open again. But if you turn the key with the door open, the bolt will activate and prevent the door from closing. The door will never close again. Copyright is turning the key with the door closed. Copyleft is turning the key with the door open. Both are the same action, both are the same thing. Both use the same mechanisms and are upheld by the same law. Only their purpose is different.

> *VI. Is the GPL CostBeneficial?*
>
> *When a software product is sold, it represents the efforts of a diverse team of individuals. The revenue from software compensates engineers, graphic artists, database programmers, hardware specialists, debuggers and a multitude of contractors, partners and vendors. In the U.S., the software sector accounted for approximately 319 million jobs in 2001 (see Appendix 8). Software development usually reflects very thin operating budgets and small margins for mistakes. Even after a software application is released, it is often not profitable until its second or third version. The developer must finance both the initial development phase and later modifications. Modifications include: bugs, enduser change requests or upgrades. In the end, successful software is the result of countless hours of programming and extensive capital outlays.*

Strangely, open source continues to thrive, even in the middle of an economical crisis. Much could be said about the viability of open source, but the facts speak for themselves.

> *Discussing the economic implications of open source, Andre Carter, President of Irimi Corporation, a technology consulting firm in Washington comments, ``The question of open source code is about whether the software developer wants to make available to the world the blueprint of what they built or simply the benefits of what they built. The notion of open source software has nothing to do with*

> *``free software.'' The purchase price of computer software is only a fraction of the total cost of ownership. So even if the price tag reads ``free'', it can end up being more expensive than software you buy. This is especially true for the typical consumer. If it requires technical knowhow to operate, doesn't offer builtin support, and demands constant attention, it won't feel free for very long.''*

In fact, this is true. But cost is actually C=A+S+O+M, where C is cost, A is acquisition cost, S is support cost, O is operational cost and M is marginal cost. And all of those must be taken into account. If you think the average life of a piece of commercial software is three years, because after that the vendor no longer supports it, then commercial software cost, C'=A'+S'+O'+M', is renewed every three years. We know that A=0, where A'>0. Now, is O+S larger or smaller than O'+S'? Does free software cost more to operate and support than commercial software? I do not think so. If open source software is not appropriate to the user, he will soon find out, and the loss he incurs in is null or very little. In turn, if the user buys commercial software only to find out it is inappropriate or too costly to operate, he already incurs in the license fee cost. Therefore, trying open source software has little risk, while trying commercial software is a risky ans expensive proposition. As far as training and other operation costs go, it has been shown that in the operating systems field, for instance, a full time Linux administrator can singlehandedly support in excess of 20 different servers, while the number for Windows servers is far lower. Windows servers require constant attention, they do not offer built-in quality support on the operational level, and yes, they do require technical know-how to operate. That is why courses such as MCSE exist, if operation and administration of Windows systems were easy, such courses would be worthless and unnecessary. The same may be said for Oracle or IBM products.

Open source software has another advantage when it comes to support services and the free market. With commercial products, any company that wishes to offer support services to users must be accordingly “allowed” by the maker of the product, All this has a cost, not only in training staff, but also a license and marketing cost. Furthermore, companies are not allowed to practice market rates for their services, since their services are now a franchise of the vendor's support service. Effectively, in some cases, a near monopoly.
With open source, the scenario is quite different. With no “mother company” to report to, each support company is free to market their support plan at whatever price they want, including as many services as they can. This effectively brings the laws of the free market into play. Better companies will do better business. Prices will come down. The user will benefit.

In short, operational and support costs for open source products, while it may be maintained that they are not much lower than for commercial products, are certainly not higher. If we add that to the actual software cost, open source becomes much more cost effective, and with a very low risk of adoption.

> *Another consideration is that support costs over the duration of a software application's availability, in some cases inevitably become the largest cost to the software developer. Customer service costs are harder to manage because consumers are rarely willing to pay for customer service, especially when they are buying lowcost software. Proprietary development firms must guarantee delivery, operation and maintenance of products.*

The laws of the market are changing. Customers are slowly adapting to a business model where software is free but services are bought. The successful operation of existing open source firms and of global companies like IBM Global Services, which has a very strong open source presence, are enough proof of this coming change.

> *Conversely, even if a firm had the best open source team on their payroll, open source inherently offers no guarantee or warranty whether it comes with a virus, back door or any other serious technical problem, and GPL open source only exacerbates these shortcomings. The significant investment of time and resources to develop software explains why developers with the hopes of recouping their investment would be unwilling to give away something that reflects years of work and investment. The developer must recoup research and development costs as well as make a profit. This uphill challenge discourages interest in the GPL, a license that stipulates that any use of GPL code mandates ``fair'' exchange of source code.*

Interestingly, the open source bandwagon seems to be growing instead of shrinking, with corporate interest rising to higher figures than ever before. Successful business models abound, and will become more prominent as the customer perspective on the software market changes.

> *Proportionality in GPL exchanges becomes an important economic consideration. If a software application representing 5000 hours uses GPL code that reflects only 100 hours, is the GPL fair in*

its argument that the entire product is GPL?

If a commercial software application representing 5000 hours of work needs to implement a function that represents 100 hours of work, they should implements the function themselves. Nobody forces anybody to use GPL code, and all parties know beforehand what the terms are.

If something is not relevant enough to consider going GPL with your code, then don't do it. If it is relevant enough for you to consider it, then is not the GPL portion indeed a core part of your work?

> *This point is of considerable concern to software companies that value their secrets, design and architecture strategies. Proponents of the GPL argue that each party in the exchange is benefiting equally, but without a means to properly make this evaluation, this position at best is overassuming.*

It is up to every participant in the exchange to assess his individual contribution and return. Many companies have assessed this, and concluded that the return obtained from the GPL codebase was large enough to provide them a greater competitive advantage than the “secrets, design and architecture strategies” that they were trying to protect by closing source.

> *The underpinning of the GPL is the argument that if you remove the existence of proprietary software, you will subsequently increase the value of a programmer's services. Programmers would be paid in an open source world, but all software will be nonproprietary and programmers will still be needed to adapt and modify products that will lead to a demand for programmers, etc. David Evans, senior vice president of NERA commenting on the tenuous nature of this dilemma wrote, ``The thousands of programmers around the world who volunteer their time for opensource projects provide the movement's greatest strengths. But one wonders how much effort these volunteers will offer to projects that are important to the future success of open source software but that are neither intellectually challenging nor of immediate utility to themselves.''*

One doesn't need to wonder. Just look at some of the open source products in existence. Furthermore, in the specific case of government software, these programmers will be employed as contractors to work upon a GPL codebase and produce custom software for government applications. This can be neither challenging nor of immediate use. But it will be paid, so it will be done, just like any piece of commercial software. But the codebase itself will be free, driving down the cost. And if what they did specificly for the government is of interest to the community, and if the government allows it, it may be reintegrated into the codebase. If it is not, they will only have the satisfaction of a job well done.

> *Hundreds of firms would have to agree on making considerable investment in open source systems, training and upgrades to their existing software. Thus, the switch would have to offer substantial and immediate cost savings, and a significant comparative advantage to staying with the status quo. Businesses today are particularly speculative about making investments based on longterm profit assumptions. Recently, hundreds of .coms relied on longterm profits to overcome exceptional short-term expenditures. Their failure has solidified this notion to be a very slippery business model indeed.*

The necessary investment is not as great as the author supposes. Most open source software can be obtained free of charge, most of it works on commodity hardware and offers better performance on this hardware than their commercial counterparts. A company that needs to upgrade their servers for performance reasons may as well change their software platform to open source, probably for the same cost in services that they would be paying for the hardware upgrade. The open source benefits are very much near term, but they do continue and scale into the long term.

> *The best ideas in technology sit on the shelf for decades. Their adoption is almost always gradual, particularly because people have to agree that the new system offers significant advantages over the old way of doing things. Regardless of the environment, people are resistant to change. While some .com firms did have technology and delivery problems, most failed because they needed a massive adoption of their ideas and concepts for shopping to occur at once. For all programmers at once to decide that all software should be free and the only thing of value would be their services is a leap in practicality and reality. The classic allegory of the prisoner's dilemma suggests that in fact programmers would be slower, particularly if they suspected their colleagues had conflicting intentions. Even in a vacuum of trust, each programmer, organization and company would be waiting for the other to forgo their current model of collecting profits to make their software open*

source.

This would be a reasonable argument, except that open source has not started yesterday. Open source has years of maturity and development, and its adoption has long ago passed its inception stage.
Secondly, the author seems attached to a near sighted notion that open source only becomes a reality when all programmers in the world turn to an open source development model. This is wrong. Both models can coexist, do coexist, have coexisted for decades.
And open source is mature enough, and has already demonstrated that it is as viable a model as the commercial closed source development model.

> *The success of an AZ open source environment would expectedly impact the software sector as a viable entity. If software is freely available, but PC's, servers and hardware maintain their value, we can only predict that the value of software companies will plummet. Hardware will come with more and more free software. Second, we can only expect that the revenues and value of the software sector will transfer to the hardware sector. Although the software sector has seen growth almost every year, it is questionable whether the GPL model will enable the software industry to continue its exceptional growth particularly when the growth in the software sector is tied to proprietary products, something the GPL is anxious to eliminate.*

The real growth in the software industry has come from custom work done for customers, implementation services, tailoring, tuning, special software, software analysis and design, in-house applications.

Commercial off-the-shelf software, the kind that would be impacted by the scenario described, plays a small part in the global software industry scene.

> *Outside of IP concerns, GPL presents a host of financial questions to corporations. While one company may decide to contribute its source code, they have no guarantee that a competitor is doing the same. Another consideration is that prevailing doubt concerning the GPL has created an environment that is actually encouraging misuse of GPL source code. If companies are concerned about any variation of these issues, it subsequently affects the value of the GPL license as well as its fair distribution of source code.*

Companies exist to create value for their stakeholders. I have somehow failed to perceive the specific devaluation in IBM stock derived from their support and adoption of open source.

> *Businesses must be concerned about the perception of the GPL. For example, experts assess the value of intellectual property when completing valuations of firms. Because GPL open source literally erases the proprietary and trade secret value of software, it can be expected that firms concerned about valuations will be very concerned about using GPL open source. This is very serious when considering that mergers, acquisitions, initial public offerings (IPO), and loans are just a few examples of critical transactions tied to the valuation a company. Even small firms that are avid users of open source to move the development of their projects fast forward, might be shunned by investors or other third parties because of their use of GPL products.*

This has no grounds. Companies are valued based on assets, yes, but aside from assets the capability of generating revenue must be valued as well. If companies were based merely on tangible assets, something like the IBM Global Services division would be worth quite little, considering all they have is people. And how would you value Oracle? Do you count the value of the database software they own as being worth nothing, since they haven't paid for it, or do you count it as infinite value, since they can issue as many licenses as they want? Company valuation is a road full of pitfalls, and it cannot be taken as lightly as the author suggests,

> *A remaining consideration is that if open source is successful, it will return goods, services and savings to consumers. Consumers in turn will be able to buy open source applications for home appliances, etc.*

This is a convoluted and strange notion. Consumers nowadays don't buy commercial software for home appliances, and there is little evidence that they will in the future, Appliances are products in-a-box, only the technical users will ever want to find out what's inside or tamper with it. And using open-source products in appliances will undoubtedly have a positive impact in lowering the cost of the appliance. Licensing commercial software for use in appliances will grow into a bigger and bigger relative expense as hardware is commoditised

and its cost decreases. The same has happened with PC's, where the cost of a commercial operating system already represents a substantial percentage of the overall consumer cost.

> *However, this theory is predicated upon consumer interest in open source products. Evans comments, ``There does not appear to be any method for the open source software method to identify and respond to the needs of nontechies...Moreover, because open source software tends to be written for use by IT professionals, there are few incentives to smooth out the rough edges and make software easier to use. And that limits the prospects for open source in the software used by vast majority of consumers. ``*

This fails to understand the changing nature of software in the modern world. Software is becoming more and more embedded, and the user will have progressively less notion of software's existence or role. In such terms, the average consumer will not be worried whether his appliance uses open source or closed source. The choice is for developers. And developers will, as we have demonstrated, benefit greatly from open source. The customer will, also, reap benefits from the usage of open source software, be it in terms of cost savings, be it in terms of interoperability, be it in terms of having more advanced software at his disposal since companies can build upon the open source codebase instead of having to re-implement every needed function every step of the way.

> *Carter continues on the vital role of the consumer suggesting, ``The notion of free software has to survive in the marketplace. Its in the marketplace that we'll discover if free software can not only address a specific defined need, but also countless undefined, unconscious needs. Those are the needs the consumer doesn't even know they have. That's a software product, a total solution. Market forces will decide if it offers that total solution to consumers.*

The notion of a “total solution” has never been proved. Consumers value choice, they value specific products, targeted to their own needs. The notion of a global, total solution for all resembles the notion of a single TV channel, or a single phone company. Monopoly restricts choice. Choice is the base of the free market. The free market is the very foundation of our economic system.

> *There are all types of consumers with ranges of needs and abilities. The guys in the lab at MIT don't need install wizards, plug and play drivers, voice based technical support and big picture manuals as part of their software. However, the elderly couple emailing their grandkids or the mother of two managing accounts on a PC in the kitchen does. With most free software when you have a problem, you may have access to some online forum to discuss that problem in a chat room with tech types, but is that of interest to the average consumer? But the problem is yours to fix, that's beyond most consumers. In the end, the free software could end up awkward and costly to the consumer. This point cannot be ignored."*

And this point will not be ignored. The advantage of support for open source products is that you can actually choose your support company based on the support it gives, instead of being locked into the specific company you bought your product from, regardless of the bad support you get from it. The market will continue to evolve, and consumers will slowly get accustomed to the notion of having software free and then paying for services. When it comes to that, we'll have a real free market for support, which is something we don't have today. And that's a good thing.

VII. GPL Open Source and the Courts

> *The acquisitiveness of the GPL inherently creates legal issues for all parties involved.*

Wrong. There are no more legal issues stemming from the GPL than might stem from any other kind of license agreement.

> *However, without legal precedent to offer definitive answers, many companies are on the sidelines awaiting the courts to provide a legal interpretation of the General Public License.*

Wrong. Legal precedent is in fact needed in order to establish the way and the extent in which the application of the GPL may be enacted, but the same holds true for any other license. Every license violation must be

ascertained by the courts. And that has never stopped people from using software.

> *Once GPL code is combined with another type of source code, the entire product is GPL.*

Wrong. The GPL applies to the distribution of modified or unmodified code. Combination of proprietary code with GPL code is legal, allowed, and requires no disclosure of either source or binary of the combined product. If you choose to distribute the combined product, then, and only then, are you subject to the GPL.

> *Subsequently, this change could occur deliberately, but it could also occur accidentally.*

Wrong. No change should ever occur accidentally in a controlled software development environment. The author seems to imply that developers could “accidentally” use GPL code in a commercial product. If that were to be the case, the same might be said about “accidental” inclusion of a worm, back door or computer virus in the said commercial product. This accidental inclusion scenario can not happen under a responsible software development process.

> *There are unlimited scenarios for accidents to occur, the license could be lost in the source code's distribution, or maybe unreadable due to a glitch in its electronic distribution.*

Wrong. Change management processes and environments exist exactly to prevent these called “accidents” from happening. The license is like any other file in the distribution. If the license could be corrupted, so could any source code file. If the electronic distribution had a “glitch”, this could easily be verified by the distribution digital checksum, and a non corrupt package could be obtained.

> *Another potentially litigious issue is whether the use of GPL tools used to manipulate code subject software to the GPL.*

Wrong. The output of GPL tools is not specifically covered by the GPL, unless the output is tightly connected with the tool itself. Even if the sentence is not very clear, the author may rest assured that the GPL does not force programs compiled with gcc to be licensed under the GPL.

> *Theoretically, a GPL tool could subject new software to GPL restrictions.*

Wrong. This is not a theoretical construct, this is a fabrication. Reading the GPL may help in clarifying these doubts. Usage of a GPL tool does not automatically subject the output of the said too to the GPL.

> *This too will have to be interpreted by a judge.*

Wrong. This is clear within the text of the GPL. If anyone claimed that the use of the GPL tool they created had the ability to apply the GPL automatically to any code it touched, then yes, it would need to be analysed by a judge. But the same might apply to any other kind of license. If my license for a word processor expires and locks me out of access to texts I have previously written while still under the license, this too will need to be interpreted by a judge. It's not a matter of being the GPL or not. It's a matter of the law of the land.

> *Regardless, unknowing users of GPL might have one intention for use of the license and find out later that it inadvertently infringed upon copyright protected work.*
> *Legal questions relevant to such an event intersect the legal arenas of intellectual property rights, contract law and liability.*

Wrong. The inadvertent use of copyrighted work constitutes a felony in itself, and will need to be appreciated by the courts as such. The fact that the user subsequently released the said work under the GPL is no different than releasing it under any other kind of license. Even with the GPL, the law of the land applies.

> *Another issue is the question of author rights. If the author decides that you are not distributing the program in a manor that meets his satisfaction, he could decide to withdraw your right to distribute the program. Again, this awaits a decision by the courts. It is not clear how the GPL exposes a licensee to this restriction.*

Wrong. It is, in fact quite clear. The GPL explains in detail the manner in which distribution is considered compliant with the GPL. If you do not comply, you are in violation of the license and therefore have lost the right to distribute. The courts will take on from there.

GPL licensees also will need clarification on this issue if they sublicense the product. Again IP, contract law and liability questions are irrelevant if a licensee violates the GPL.

Wrong. The GPL is quite clear in itself.
I believe the author may have wanted to say that the questions were “relevant” if you violate the GPL. In fact, they are, as they would be in case you violated any other license. Again, it's not the GPL that cause it. It's the law. And to be against it, is to be against the law.

Section 2 of the GPL says that identifiable sections of a work that are not derived from a GPL program and that ``can be reasonably considered independent and separate'' are not subject to the GPL when distributed as separate works. This becomes a very subjective issue as well. It is questionable whether an owner could accurately identify a work as under the original GPL code or otherwise.

Wrong. It is not questionable at all. The work may be compared by specialists, and they will inform the courts whether they believe the work is independent and separate or not. Again, the seemingly “subjective” issue arises from the way democratic countries have chosen to impose justice and laws across the land.

Section 2 also says that a ``mere aggregation of another work not based on the [GPL] Program on a volume of a storage or distribution medium does not bring the other work under the scope of this License.'' This issue also leads to legal questions, forcing licensees to have to decide (to the best of their ability) whether combined products work as a GPL or nonGPL product.

Wrong. This issue is, in fact, one of the simplest. You just have to read the text.

Use of GPL code that includes code licensed from a third party could obligate you to sublicense the other party's code under the GPL. In other words, if you buy a bucket of apples under a restrictive agreement that happens to include a couple oranges that you like, it is questionable what rights you have to buy (or sell) oranges you get directly from the supplier once the previous agreement is in place.

Wrong. This code you receive under the GPL is licensed to you under the GPL. Only if you change it redistribute it you will need to license the changed code under the GPL. There is specific work done on the subject of “license compatibility” between the GPL and other available licenses. Some allow you to relicense the bundle under the GPL, others don't. If they don't, you cannot mix the code. It is very simple. You just have to read.

Likewise, it is questionable if the owner of code that is GPL'ed can change his mind and distribute it with a new license.

Wrong. This is not only “not questionable” but also “already done”. The original owner of the code can license it separately under any other license he wishes to, and the recipient of the code must abide only by the license under which he got the code. What the owner cannot do is revoke the GPL license he granted.

Each of these scenarios questions the ability of GPL licenses to follow source code that becomes GPL.
Inevitable jurisprudence will tackle these and other gray areas of the GPL.

Right. Yes it will. That's what jurisprudence was invented for.

VIII. Conclusion

Open source as a development model is helpful to the software industry. For example, software distributed under the BSD license is very popular. The BSD license (see Appendix 9) enables companies, independent developers and the academic community to fluidly exchange software source code. In addition, developers frequently create their own versions of open source licenses to independently distribute their source code.

Yes, the open source model is quite helpful to the software industry. But more than that, the open source model and the people and companies that follow it ARE the software industry, they are part of the software industry. This is not an “us and them” situation. This is about a new model of doing things, so that all may reap the most benefit from the work of all.

> *The GPL's resistance to commonplace exchange of open source and proprietary has the potential to negatively impact the research and development budgets of companies. IBM, Sun and Microsoft alone combine for over ten billion dollars spent annually on research and development. It stands to reason that if the ownership of intellectual property is affected, dollars spent on research and development would be at risk as well. Removing the economic incentive for firms to own the rights to products spawned from research and development programs is the surest way to end their existence.*

There are no grounds on which to base this conclusion. Many companies have joined the open source community and have since made no significant cuts in their R&D budget that can be ascribed to that. The government has a long tradition of funding publicly available research, and this research has done more for the advancement of science and technology than most closed source initiatives. The move to open source by software companies will most likely be in the area of server software, development kits and API's and commodity software. None of these areas will benefit much more from the companies R&D budget than it will benefit from the joint effort of the open source community's R&D. This will, in fact, leave companies with more money to invest in fundamental research and exploring new avenues for their proprietary products. And this, yes, this, not R&D in Office technologies, will drive the industry further and create benefits for consumers.

> *The GPL has many risks, but the greatest is its threat to the cooperation between different parties who collaborate and create new technologies. Today, government, commercial enterprise, academicians, etc. have a system to converge. Conversely, the GPL represents divergence; proposing to remove the current infrastructure of intellectual property rights, enforceable protection and economic incentive.*

The GPL poses no risk to collaboration, in fact, it does the exact opposite. The cooperation between parties is based on disclosure, and the GPL provides a common ground for this. The government. The academic community, the independent developer community, the commercial open source companies, all of this vast body of entities have found collaboration possible, productive and stimulant under the GPL. The only entities that have yet failed to find a way in which they can contribute and participate are the commercial closed source software companies. The open source community has shown the closed source community many different ways in which this collaboration may be attained. But the action lies within the closed source community. It is the closed source companies that have to choose whether they wish to take part in the global open source initiative. Some have already started doing it, with very good results. The others are invited to follow academia, other companies, the government and the community into this new way of building software and sharing knowledge for the common advancement of all.

> *While GPL advocates are quite active in their promotion of copyleft, few would disagree that its widespread adoption would present a radical change to an industry sector responsible for almost 350 billion dollars in sales annually worldwide (see Appendix 10). Open source can coexist with the status quo, but GPL cannot coexist with traditional open source or proprietary source code.*

This is a wrong assumption that seems totally out of place in any serious study on the economic and social impacts of open source. The GPL can coexist with every other development model in the world. In fact, it has done so for the last 15 years.

> *Considering the varied implications of the use of open source its, impact on the future of technology clearly deserves further study.*

Indeed it does. But certainly it deserves to be analysed from an informed perspective, with no bias, and supported by people from both sides of the court.

In fact, this would be an excellent opportunity for the government to fund private research into the benefits and potential issues arising with the wide scale adoption of open source and GPL code by government institutions. The study may be funded with substantially less money than the annual licensing fees the government pays for closed source desktop products in any single one of its many departments.

Important considerations include:
1 Engineering software has become considerably complicated and rigorous. It is not unusual for software to include millions of lines of source code. If the incentive to develop software is changed, we can subsequently expect the quality and efficiency of software to change.

In fact software has grown both in size and in complexity. And if the incentives to software production change, the quality of software may change too.
If software becomes more and more open to scrutiny and improvement, the overall quality of software will definitively rise.
If the effort of companies goes towards improving existing code instead of rewriting many different proprietary versions of the same kind of software, the quality and functionality of software will definitively rise.
If the financial benefits model from software commercialization changes, the industry will, of course, adapt, as it always has done in the past, and this will not have a specific impact on any of the characteristics of its products. A more expensive product is not always a better product, as most consumers know.

2 There remains considerable differences within the GPL open source community. It is questionable whether these groups will continue to be proponents of the GPL in its current form or opt for changes in the immediate future.

Many different open source licenses exist. The GPL has been chosen by the majority of open source development teams because of the openness guarantees it provides, and because it prevents others from unlawfully appropriating their intellectual property. The GPL insures that what is made open stays open, and that software continues to exist as a transparent entity in which the individual contributions from either party remain measurable and understandable.

If any groups within the open source community choose to change their licensing model for software, this will not have an impact on the community as a whole. If a better license comes along, the developers will use it, according to their needs and their wishes. The GPL is not a company, the GPL does not have a stock market price that goes up or down if people join or leave its ranks.

It is merely a tool, and so far it has been the best tool to provide a legal framework for developers to ascertain their wish: the openness of code.

3 Open source has successfully been used in proprietary software. In addition, academic and government projects have been successful particularly because of commercial interest. Private enterprise offers unique efficiencies for the success of government funded research.

Commercial interest will always exist. Government funded research will always exist. And it will be commissioned into the private sector, either with the GPL or without it. The GPL does not end the government investment on R&D. Actually, it may well be the contrary, as research may be done on top of a rich and available codebase.

Private companies will still be the ones doing the research. The single difference with the GPL is that after the research is completed, they will have to leverage their know-how in order to make additional profits from the government money, instead of selling closed source products whose research was paid for with taxpayer money. This is, we believe, a good thing.

4 Open source GPL use by government agencies could easily become a national security concern. Government use of software in the public domain is exceptionally risky.

This is totally wrong. Security cannot be obtained through obscurity alone.
And closed source software has already, in itself, created significant risks to the security of government institutions, be it by back doors, misconfiguration, bugs, holes, patches and fixes that take too long to be published by commercial companies because they do not bring direct revenue.
To an open source company, a patch is a matter of code quality, it is a matter of peer pressure, of peer recognition, it is a matter that ties directly with the company image, And in open source, the company's reputation is the main source of its income.
To a closed source company, software sales are their main source of revenue. Patches do not bring income. The drive must be getting the customers to upgrade and pay license fees, not to keep them happy with the version they own.

Thus, providing security and stability for current products is an objective of the open source model, and contrary to the motivations of the closed source companies. This, in fact, makes open source software inherently more secure than closed source software.

As to the issue of disclosure, the GPL does not mandate disclosure unless there is redistribution. The government is not a software company. The government need not redistribute his code. Therefore, the government need not disclose source. The whole issue of GPL as a security risk has no grounds, and is purely the result of uninformed comments by companies whose vested interest is in keeping source closed and making money out of new releases.

> *5 Reverse engineering, perpetuated by GPL proponents, threatens not only the owners of intellectual property, but also the software industry itself.*

This is false. Reverse engineering for interoperability is a legal and accepted practice, and an advanced scientific discipline of the computer science curricula. Its use has been validated both by law and by practice, and it stands as the base of a wide number of industries even outside the software world.
Reverse engineering happens because of lack of disclosure of interoperability information. This lack of disclosure happens because of greed, forcing consumers into monopolistic lock-ins.
Interoperability, enabled by reverse engineering, allows for choice, Choice is the basis of the free market. And the free market is the cornerstone of our society.

> *6 Use of GPL open source creates a number of economic concerns for firms. For example, the valuation of a software company could be significantly effected if it uses source code licensed under the GPL for the development of its products.*

This applies to software companies, which represent a small part of the whole software industry and of the economy as a whole. Nevertheless, the valuation of a software company should be based on the value it can produce, not on the assets it sits on. For a software company, valuation should be based on the revenue it can obtain from its products and services. This revenue has nothing to do with either its products are open sourced or not.

For every other company in the economy, the value of open source is quite positive. It reduces cost. It promotes flexibility. If creates diversity in software offerings, by promoting interoperability. By driving IT costs down, and crating a free market in which to select products and suppliers, open source has a net positive impact on every company's balance sheet, therefore increasing its market value by any standard.

As for software companies, the decision to base their software on open source must be accompanied by a shift in their business model. They may either create a business model that will lead consumers to buy from them instead of others, or they may choose to base their model around support or value added services. The world is changing, as it has changed so many times in the past. The successful companies adapt. The others suffer from their lack of flexibility.

> *7 The courts have yet to weigh in on the General Public License. Without legal interpretation, the use of the GPL could be perilous to users in a number of scenarios.*

The enforceability of software licenses, as a whole, is still an open issue with the courts. This is not specifically related to the GPL, or to any other license, including those that allow for inclusion of source code in commercial products. The courts will continue to do their duty of interpreting law and private contracts, and providing decisions and guidelines to interpret subsequent cases.

But the GPL does not expose the licensee to any dangerous scenario. In fact, the GPL grants far more right than any other license, namely those for closed source products.

As far as the user is concerned, the GPL poses no risk whatsoever to the user. The GPL does not even apply to the user. The GPL covers modifications and redistribution of code, not usage. Stating that it may cause risks to the user has no grounds whatsoever, insofar as it does not even apply to them.

Specifically in the case of government use, and applicable to any private non-software company, the GPL is the only license that fully grants them complete and unaltered rights to own and inspect the source code for their products, either contract the maintenance and support of said products or provide internal facilities for that

purpose, and still carry no weight of necessity to disclose, since the products will not be subject to any redistribution to third parties.

In short, this is a changing world. The concept of software is changing, shifting power from a few companies into the hands of consumers and end users. This will have, like any liberalization, a positive effect on the economy. Freed from the monopoly-like control of closed source companies, enforced by proprietary formats, proprietary interfaces, proprietary requisites, both companies and government will be able to access state-of-the-art software at reasonable costs, benefiting from a wide pool of skilled resources to provide support and customization.

The only ones with anything to lose will be the ones who close their eyes and resist this change. The flexible companies will adapt and evolve. The other will suffer the consequences, like every time evolution occurs.