First Monday, Volume 4, Number 8 - 2 August 1999

f i ® s t m ¤ ñ d @ ¥
PEER-REVIEWED JOURNAL ON THE INTERNET

Anarchism Triumphant: Free Software and the Death of Copyright

by Eben Moglen

The spread of the Linux operating system kernel has directed attention at the free software movement. This paper shows why free software, far from being a marginal participant in the commercial software market, is the vital first step in the withering away of the intellectual property system.

Contents

I. Software as Property: The Theoretical Paradox

SOFTWARE: no other word so thoroughly connotes the practical and social effects of the digital revolution. Originally, the term was purely technical, and denoted the parts of a computer system that, unlike "hardware," which was unchangeably manufactured in system electronics, could be altered freely. The first software amounted to the plug configuration of cables or switches on the outside panels of an electronic device, but as soon as linguistic means of altering computer behavior had been developed, "software" mostly denoted the expressions in more or less human-readable language that both described and controlled machine behavior [1].

That was then and this is now. Technology based on the manipulation of digitally-encoded information is now socially dominant in most aspects of human culture in the "developed" societies [2]. The movement from analog to digital representation - in video, music, printing, telecommunications, and even choreography, religious worship, and sexual gratification - potentially turns all forms of human symbolic activity into software, that is, modifiable instructions for describing and controlling the behavior of machines. By a conceptual back-formation characteristic of Western scientistic thinking, the division between hardware and software is now being observed in the natural or social world, and has become a new way to express the conflict between ideas of determinism and free will, nature and nurture, or genes and culture. Our "hardware," genetically wired, is our nature, and determines us. Our nurture is "software," establishes our cultural programming, which is our comparative freedom. And so on, for those reckless of blather [3]. Thus "software" becomes a viable metaphor for all symbolic activity, apparently divorced from the technical context of the word's origin, despite the unease raised in the technically competent when the term is thus bandied about, eliding the conceptual significance of its derivation [4].

But the widespread adoption of digital technology for use by those who do not understand the principles of its operation, while it apparently licenses the broad metaphoric employment of "software," does not in fact permit us to ignore the computers that are now everywhere underneath our social skin. The movement from analog to digital is more important for the structure of social and legal relations than the more famous if less certain movement from status to contract [5]. This is bad news for those legal thinkers who do not understand it, which is why so much pretending to understand now goes so floridly on. Potentially, however, our great transition is very good news for those who can turn this new-found land into property for themselves. Which is why the current "owners" of software so strongly support and encourage the ignorance of everyone else. Unfortunately for them - for reasons familiar to legal theorists who haven't yet understood how to apply their traditional logic in this area - the trick won't work. This paper explains why [6].

We need to begin by considering the technical essence of the familiar devices that surround us in the era of "cultural software." A CD player is a good example. Its primary input is a bitstream read from an optical storage disk. The bitstream describes music in terms of measurements, taken 44,000 times per second, of frequency and amplitude in each of two audio channels. The player's primary output is analog audio signals [7]. Like everything else in the digital world, music as seen by a CD player is mere numeric information; a particular recording of Beethoven's Ninth Symphony recorded by Arturo Toscanini and the NBC Symphony Orchestra and Chorale is (to drop a few insignificant digits) 1276749873424, while Glenn Gould's peculiarly perverse last recording of the Goldberg Variations is (similarly rather truncated) 767459083268.

Oddly enough, these two numbers are "copyrighted." This means, supposedly, that you can't possess another copy of these numbers, once fixed in any physical form, unless you have licensed them. And you can't turn 767459083268 into

2347895697 for your friends (thus correcting Gould's ridiculous judgment about tempi) without making a "derivative work," for which a license is necessary.

At the same time, a similar optical storage disk contains another number, let us call it 7537489532. This one is an algorithm for linear programming of large systems with multiple constraints, useful for example if you want to make optimal use of your rolling stock in running a freight railroad. This number (in the U.S.) is "patented," which means you cannot derive 7537489532 for yourself, or otherwise "practice the art" of the patent with respect to solving linear programming problems no matter how you came by the idea, including finding it out for yourself, unless you have a license from the number's owner.

Then there's 9892454959483. This one is the source code for Microsoft Word. In addition to being "copyrighted," this one is a trade secret. That means if you take this number from Microsoft and give it to anyone else you can be punished.

Lastly, there's 588832161316. It doesn't do anything, it's just the square of 767354. As far as I know, it isn't owned by anybody under any of these rubrics. Yet.

At this point we must deal with our first objection from the learned. It comes from a creature known as the IPdroid. The droid has a sophisticated mind and a cultured life. It appreciates very much the elegant dinners at academic and ministerial conferences about the TRIPs, not to mention the privilege of frequent appearances on MSNBC. It wants you to know that I'm committing the mistake of confusing the embodiment with the intellectual property itself. It's not the number that's patented, stupid, just the Kammerkar algorithm. The number *can* be copyrighted, because copyright covers the expressive qualities of a particular tangible embodiment of an idea (in which some functional properties may be mysteriously merged, provided that they're not too merged), but not the algorithm. Whereas the number isn't patentable, just the "teaching" of the number with respect to making railroads run on time. And the number representing the source code of Microsoft Word can be a trade secret, but if you find it out for yourself (by performing arithmetic manipulation of other numbers issued by Microsoft, for example, which is known as "reverse engineering"), you're not going to be punished, at least if you live in some parts of the United States.

This droid, like other droids, is often right. The condition of being a droid is to know everything about something and nothing about anything else. By its timely and urgent intervention the droid has established that the current intellectual property system contains many intricate and ingenious features. The complexities combine to allow professors to be erudite, Congressmen to get campaign contributions, lawyers to wear nice suits and tassel loafers, and Murdoch to be rich. The complexities mostly evolved in an age of industrial information distribution, when information was inscribed in analog forms on physical objects that cost something significant to make, move, and sell. When applied to digital information that moves frictionlessly through the network and has zero marginal cost per copy, everything still works, mostly, as long as you don't stop squinting.

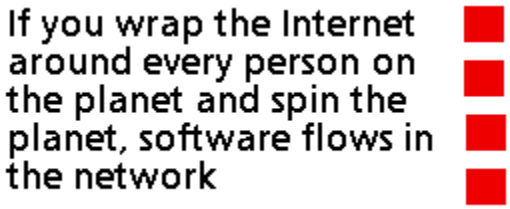
But that wasn't what I was arguing about. I wanted to point out something else: that our world consists increasingly of nothing but large numbers (also known as bitstreams), and that - for reasons having nothing to do with emergent properties of the numbers themselves - the legal system is presently committed to treating similar numbers radically differently. No one can tell, simply by looking at a number that is 100 million digits long, whether that number is subject to patent, copyright, or trade secret protection, or indeed whether it is "owned" by anyone at all. So the legal system we have - blessed as we are by its consequences if we are copyright teachers, Congressmen, Gucci-gulchers or Big Rupert himself - is compelled to treat indistinguishable things in unlike ways.

Now, in my role as a legal historian concerned with the secular (that is, very long term) development of legal thought, I claim that legal regimes based on sharp but unpredictable distinctions among similar objects are radically unstable. They fall apart over time because every instance of the rules' application is an invitation to at least one side to claim that instead of fitting in ideal category A the particular object in dispute should be deemed to fit instead in category B, where the rules will be more favorable to the party making the claim. This game - about whether a typewriter should be deemed a musical instrument for purposes of railway rate regulation, or whether a steam shovel is a motor vehicle - is the frequent stuff of legal ingenuity. But when the conventionally-approved legal categories require judges to distinguish among the identical, the game is infinitely lengthy, infinitely costly, and almost infinitely offensive to the unbiased bystander [8].

Thus parties can spend all the money they want on all the legislators and judges they can afford - which for the new "owners" of the digital world is quite a few - but the rules they buy aren't going to work in the end. Sooner or later, the paradigms are going to collapse. Of course, if later means two generations from now, the distribution of wealth and power sanctified in the meantime may not be reversible by any course less drastic than a *bellum servile* of couch potatoes against media magnates. So knowing that history isn't on Bill Gates' side isn't enough. We are predicting the future in a very limited sense: we know that the existing rules, which have yet the fervor of conventional belief solidly enlisted behind them, are no longer meaningful. Parties will use and abuse them freely until the mainstream of "respectable" conservative opinion acknowledges their death, with uncertain results. But realistic scholarship should already be turning its attention to the clear need for new thoughtways.

When we reach this point in the argument, we find ourselves contending with the other primary protagonist of educated idiocy: the econodwarf. Like the IPdroid, the econodwarf is a species of hedgehog,[9] but where the droid is committed to logic over experience, the econodwarf specializes in an energetic and well-focused but entirely erroneous view of human nature. According to the econodwarf's vision, each human being is an individual possessing "incentives," which can be retrospectively unearthed by imagining the state of the bank account at various times. So in this instance the econodwarf feels compelled to object that without the rules I am lampooning, there would be no incentive to create the things the rules treat as property: without the ability to exclude others from music there would be no music, because no one could be sure of getting paid for creating it.

Music is not really our subject; the software I am considering at the moment is the old kind: computer programs. But as he is determined to deal at least cursorily with the subject, and because, as we have seen, it is no longer really possible to distinguish computer programs from music performances, a word or two should be said. At least we can have the satisfaction of indulging in an argument *ad pygmeam*. When the econodwarf grows rich, in my experience, he attends the opera. But no matter how often he hears *Don Giovanni* it never occurs to him that Mozart's fate should, on his logic, have entirely discouraged Beethoven, or that we have *The Magic Flute* even though Mozart knew very well he wouldn't be paid. In fact, *The Magic Flute*, *St. Matthew's Passion*, and the motets of the wife-murderer Carlo Gesualdo are all part of the centuries-long tradition of free software, in the more general sense, which the econodwarf never quite acknowledges.



The dwarf's basic problem is that "incentives" is merely a metaphor, and as a metaphor to describe human creative activity it's pretty crummy. I have said this before,[10] but the better metaphor arose on the day Michael Faraday first noticed what happened when he wrapped a coil of wire around a magnet and spun the magnet. Current flows in such a wire, but we don't ask what the incentive is for the electrons to leave home. We say that the current results from an emergent property of the system, which we call induction. The question we ask is "what's the resistance of the wire?" So Moglen's Metaphorical Corollary to Faraday's Law says that if you wrap the Internet around every person on the planet and spin the planet, software flows in the network. It's an emergent property of connected human minds that they create things for one another's pleasure and to conquer their uneasy sense of being too alone. The only question to ask is, what's the resistance of the network? Moglen's Metaphorical Corollary to Ohm's Law states that the resistance of the network is directly proportional to the field strength of the "intellectual property" system. So the right answer to the econodwarf is, resist the resistance.

Of course, this is all very well in theory. "Resist the resistance" sounds good, but we'd have a serious problem, theory notwithstanding, if the dwarf were right and we found ourselves under-producing good software because we didn't let people own it. But dwarves and droids are formalists of different kinds, and the advantage of realism is that if you start from the facts the facts are always on your side. It turns out that treating software as property makes bad software.

II. Software as Property: The Practical Problem

In order to understand why turning software into property produces bad software, we need an introduction to the history of the art. In fact, we'd better start with the word "art" itself. The programming of computers combines determinate reasoning with literary invention.

At first glance, to be sure, source code appears to be a non-literary form of composition [11]. The primary desideratum in a computer program is that it works, that is to say, performs according to specifications formally describing its outputs in terms of its inputs. At this level of generality, the functional content of programs is all that can be seen.

But working computer programs exist as parts of computer systems, which are interacting collections of hardware, software, and human beings. The human components of a computer system include not only the users, but also the (potentially different) persons who maintain and improve the system. Source code not only communicates with the computer that executes the program, through the intermediary of the compiler that produces machine-language object code, but also with other programmers.

The function of source code in relation to other human beings is not widely grasped by non-programmers, who tend to think of computer programs as incomprehensible. They would be surprised to learn that the bulk of information contained in most programs is, from the point of view of the compiler or other language processor, "comment," that is, non-functional material. The comments, of course, are addressed to others who may need to fix a problem or to alter or enhance the program's operation. In most programming languages, far more space is spent in telling people what the program does than in telling the computer how to do it.

The design of programming languages has always proceeded under the dual requirements of complete specification for machine execution and informative description for human readers. One might identify three basic strategies in language design for approaching this dual purpose. The first, pursued initially with respect to the design of languages specific to particular hardware products and collectively known as "assemblers," essentially separated the human- and machine-communication portions of the program. Assembler instructions are very close relatives of machine-language instructions: in general, one line of an assembler program corresponds to one instruction in the native language of the machine. The programmer controls machine operation at the most specific possible level, and (if well-disciplined) engages in running commentary alongside the machine instructions, pausing every few hundred instructions to create "block comments," which provide a summary of the strategy of the program, or document the major data structures the program manipulates.

A second approach, characteristically depicted by the language COBOL (which stood for "Common Business-Oriented Language"), was to make the program itself look like a set of natural language directions, written in a crabbed but

theoretically human-readable style. A line of COBOL code might say, for example "MULTIPLY PRICE TIMES QUANTITY GIVING EXPANSION." At first, when the Pentagon and industry experts began the joint design of COBOL in the early 1960's, this seemed a promising approach. COBOL programs appeared largely self-documenting, allowing both the development of work teams able to collaborate on the creation of large programs, and the training of programmers who, while specialized workers, would not need to understand the machine as intimately as assembler programs had to. But the level of generality at which such programs documented themselves was wrongly selected. A more formulaic and compressed expression of operational detail "expansion = price x quantity," for example, was better suited even to business and financial applications where the readers and writers of programs were accustomed to mathematical expression, while the processes of describing both data structures and the larger operational context of the program were not rendered unnecessary by the wordiness of the language in which the details of execution were specified.

Accordingly, language designers by the late 1960s began experimenting with forms of expression in which the blending of operational details and non-functional information necessary for modification or repair was more subtle. Some designers chose the path of highly symbolic and compressed languages, in which the programmer manipulated data abstractly, so that "A x B" might mean the multiplication of two integers, two complex numbers, two vast arrays, or any other data type capable of some process called "multiplication," to be undertaken by the computer on the basis of the context for the variables "A" and "B" at the moment of execution [12]. Because this approach resulted in extremely concise programs, it was thought, the problem of making code comprehensible to those who would later seek to modify or repair it was simplified. By hiding the technical detail of computer operation and emphasizing the algorithm, languages could be devised that were better than English or other natural languages for the expression of stepwise processes. Commentary would be not only unnecessary but distracting, just as the metaphors used to convey mathematical concepts in English do more to confuse than to enlighten.

How We Created the Microbrain Mess

Thus the history of programming languages directly reflected the need to find forms of human-machine communication that were also effective in conveying complex ideas to human readers. "Expressivity" became a property of programming languages, not because it facilitated computation, but because it facilitated the collaborative creation and maintenance of increasingly complex software systems.

At first impression, this seems to justify the application of traditional copyright thinking to the resulting works. Though substantially involving "functional" elements, computer programs contained "expressive" features of paramount importance. Copyright doctrine recognized the merger of function and expression as characteristic of many kinds of copyrighted works. "Source code," containing both the machine instructions necessary for functional operation and the expressive "commentary" intended for human readers, was an appropriate candidate for copyright treatment.

True, so long as it is understood that the expressive component of software was present solely in order to facilitate the making of "derivative works." Were it not for the intention to facilitate alteration, the expressive elements of programs would be entirely supererogatory, and source code would be no more copyrightable than object code, the output of the language processor, purged of all but the program's functional characteristics.

The state of the computer industry throughout the 1960's and 1970's, when the grundnorms of sophisticated computer programming were established, concealed the tension implicit in this situation. In that period, hardware was expensive. Computers were increasingly large and complex collections of machines, and the business of designing and building such an array of machines for general use was dominated, not to say monopolized, by one firm. IBM gave away its software. To be sure, it owned the programs its employees wrote, and it copyrighted the source code. But it also distributed the programs - including the source code - to its customers at no additional charge, and encouraged them to make and share improvements or adaptations of the programs thus distributed. For a dominant hardware manufacturer, this strategy made sense: better programs sold more computers, which is where the profitability of the business rested.

Computers, in this period, tended to aggregate within particular organizations, but not to communicate broadly with one another. The software needed to operate was distributed not through a network, but on spools of magnetic tape. This distribution system tended to centralize software development, so that while IBM customers were free to make modifications and improvements to programs, those modifications were shared in the first instance with IBM, which then considered whether and in what way to incorporate those changes in the centrally-developed and distributed version of the software. Thus in two important senses the best computer software in the world was free: it cost nothing to acquire, and the terms on which it was furnished both allowed and encouraged experimentation, change, and improvement [13]. That the software in question was IBM's property under prevailing copyright law certainly established some theoretical limits on users' ability to distribute their improvements or adaptations to others, but in practice mainframe software was cooperatively developed by the dominant hardware manufacturer and its technically-sophisticated users, employing the manufacturer's distribution resources to propagate the resulting improvements through the user community. The right to exclude others, one of the most important "sticks in the bundle" of property rights (in an image beloved of the United States Supreme Court), was practically unimportant, or even undesirable, at the heart of the software business [14].

After 1980, everything was different. The world of mainframe hardware gave way within ten years to the world of the commodity PC. And, as a contingency of the industry's development, the single most important element of the software running on that commodity PC, the operating system, became the sole significant product of a company that made no hardware. High-quality basic software ceased to be part of the product-differentiation strategy of hardware manufacturers. Instead, a firm with an overwhelming share of the market, and with the near-monopolist's ordinary absence of interest in fostering diversity, set the practices of the software industry. In such a context, the right to exclude others from participation in the product's formation became profoundly important. Microsoft's power in the market rested entirely on its

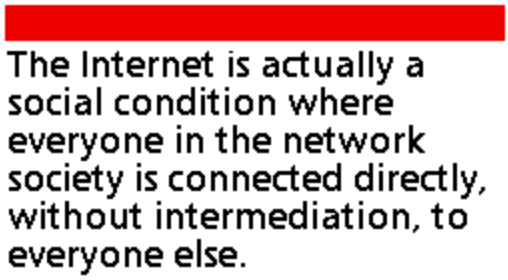
ownership of the Windows source code.

To Microsoft, others' making of "derivative works," otherwise known as repairs and improvements, threatened the central asset of the business. Indeed, as subsequent judicial proceedings have tended to establish, Microsoft's strategy as a business was to find innovative ideas elsewhere in the software marketplace, buy them up and either suppress them or incorporate them in its proprietary product. The maintenance of control over the basic operation of computers manufactured, sold, possessed, and used by others represented profound and profitable leverage over the development of the culture [15]; the right to exclude returned to center stage in the concept of software as property.

The result, so far as the quality of software was concerned, was disastrous. The monopoly was a wealthy and powerful corporation that employed a large number of programmers, but it could not possibly afford the number of testers, designers, and developers required to produce flexible, robust and technically-innovative software appropriate to the vast array of conditions under which increasingly ubiquitous personal computers operated. Its fundamental marketing strategy involved designing its product for the least technically-sophisticated users, and using "fear, uncertainty, and doubt" (known within Microsoft as "FUD") to drive sophisticated users away from potential competitors, whose long-term survivability in the face of Microsoft's market power was always in question.

Without the constant interaction between users able to repair and improve and the operating system's manufacturer, the inevitable deterioration of quality could not be arrested. But because the personal computer revolution expanded the number of users exponentially, almost everyone who came in contact with the resulting systems had nothing against which to compare them. Unaware of the standards of stability, reliability, maintainability and effectiveness that had previously been established in the mainframe world, users of personal computers could hardly be expected to understand how badly, in relative terms, the monopoly's software functioned. As the power and capacity of personal computers expanded rapidly, the defects of the software were rendered less obvious amidst the general increase of productivity. Ordinary users, more than half afraid of the technology they almost completely did not understand, actually welcomed the defectiveness of the software. In an economy undergoing mysterious transformations, with the concomitant destabilization of millions of careers, it was tranquilizing, in a perverse way, that no personal computer seemed to be able to run for more than a few consecutive hours without crashing. Although it was frustrating to lose work in progress each time an unnecessary failure occurred, the evident fallibility of computers was intrinsically reassuring [16].

None of this was necessary. The low quality of personal computer software could have been reversed by including users directly in the inherently evolutionary process of software design and implementation. A Lamarckian mode, in which improvements could be made anywhere, by anyone, and inherited by everyone else, would have wiped out the deficit, restoring to the world of the PC the stability and reliability of the software made in the quasi-propertarian environment of the mainframe era. But the Microsoft business model precluded Lamarckian inheritance of software improvements. Copyright doctrine, in general and as it applies to software in particular, biases the world towards creationism; in this instance, the problem is that BillG the Creator was far from infallible, and in fact he wasn't even trying.



To make the irony more severe, the growth of the network rendered the non-propertarian alternative even more practical. What scholarly and popular writing alike denominate as a thing ("the Internet") is actually the name of a social condition: the fact that everyone in the network society is connected directly, without intermediation, to everyone else [17]. The global interconnection of networks eliminated the bottleneck that had required a centralized software manufacturer to rationalize and distribute the outcome of individual innovation in the era of the mainframe.

And so, in one of history's little ironies, the global triumph of bad software in the age of the PC was reversed by a surprising combination of forces: the social transformation initiated by the network, a long-discarded European theory of political economy, and a small band of programmers throughout the world mobilized by a single simple idea.

Software Wants to Be Free; or, How We Stopped Worrying and Learned to Love the Bomb

Long before the network of networks was a practical reality, even before it was an aspiration, there was a desire for computers to operate on the basis of software freely available to everyone. This began as a reaction against propertarian software in the mainframe era, and requires another brief historical digression.

Even though IBM was the largest seller of general purpose computers in the mainframe era, it was not the largest designer and builder of such hardware. The telephone monopoly, American Telephone & Telegraph, was in fact larger

than IBM, but it consumed its products internally. And at the famous Bell Labs research arm of the telephone monopoly, in the late 1960's, the developments in computer languages previously described gave birth to an operating system called Unix.

The idea of Unix was to create a single, scalable operating system to exist on all the computers, from small to large, that the telephone monopoly made for itself. To achieve this goal meant writing an operating system not in machine language, nor in an assembler whose linguistic form was integral to a particular hardware design, but in a more expressive and generalized language. The one chosen was also a Bell Labs invention, called "C" [18]. The C language became common, even dominant, for many kinds of programming tasks, and by the late 1970's the Unix operating system written in that language had been transferred (or "ported," in professional jargon) to computers made by many manufacturers and of many designs.

AT&T distributed Unix widely, and because of the very design of the operating system, it had to make that distribution in C source code. But AT&T retained ownership of the source code and compelled users to purchase licenses that prohibited redistribution and the making of derivative works. Large computing centers, whether industrial or academic, could afford to purchase such licenses, but individuals could not, while the license restrictions prevented the community of programmers who used Unix from improving it in an evolutionary rather than episodic fashion. And as programmers throughout the world began to aspire to and even expect a personal computer revolution, the "unfree" status of Unix became a source of concern.

Between 1981 and 1984, one man envisioned a crusade to change the situation. Richard M. Stallman, then an employee of MIT's Artificial Intelligence Laboratory, conceived the project of independent, collaborative redesign and implementation of an operating system that would be true free software. In Stallman's phrase, free software would be a matter of freedom, not of price. Anyone could freely modify and redistribute such software, or sell it, subject only to the restriction that he not try to reduce the rights of others to whom he passed it along. In this way free software could become a self-organizing project, in which no innovation would be lost through proprietary exercises of rights. The system, Stallman decided, would be called GNU, which stood (in an initial example of a taste for recursive acronyms that has characterized free software ever since), for "GNU's Not Unix." Despite misgivings about the fundamental design of Unix, as well as its terms of distribution, GNU was intended to benefit from the wide if unfree source distribution of Unix. Stallman began Project GNU by writing components of the eventual system that were also designed to work without modification on existing Unix systems. Development of the GNU tools could thus proceed directly in the environment of university and other advanced computing centers around the world.

The scale of such a project was immense. Somehow, volunteer programmers had to be found, organized, and set to work building all the tools that would be necessary for the ultimate construction. Stallman himself was the primary author of several fundamental tools. Others were contributed by small or large teams of programmers elsewhere, and assigned to Stallman's project or distributed directly. A few locations around the developing network became archives for the source code of these GNU components, and throughout the 1980's the GNU tools gained recognition and acceptance by Unix users throughout the world. The stability, reliability, and maintainability of the GNU tools became a by-word, while Stallman's profound abilities as a designer continued to outpace, and provide goals for, the evolving process. The award to Stallman of a MacArthur Fellowship in 1990 was an appropriate recognition of his conceptual and technical innovations and their social consequences.

Project GNU, and the Free Software Foundation to which it gave birth in 1985, were not the only source of free software ideas. Several forms of copyright license designed to foster free or partially free software began to develop in the academic community, mostly around the Unix environment. The University of California at Berkeley began the design and implementation of another version of Unix for free distribution in the academic community. BSD Unix, as it came to be known, also treated AT&T's Unix as a design standard. The code was broadly released and constituted a reservoir of tools and techniques, but its license terms limited the range of its application, while the elimination of hardware-specific proprietary code from the distribution meant that no one could actually build a working operating system for any particular computer from BSD. Other university-based work also eventuated in quasi-free software; the graphical user interface (or GUI) for Unix systems called X Windows, for example, was created at MIT and distributed with source code on terms permitting free modification. And in 1989-1990, an undergraduate computer science student at the University of Helsinki, Linus Torvalds, began the project that completed the circuit and fully energized the free software vision.

What Torvalds did was to begin adapting a computer science teaching tool for real life use. Andrew Tannenbaum's MINIX kernel [19], was a staple of Operating Systems courses, providing an example of basic solutions to basic problems. Slowly, and at first without recognizing the intention, Linus began turning the MINIX kernel into an actual kernel for Unix on the Intel x86 processors, the engines that run the world's commodity PCs. As Linus began developing this kernel, which he named Linux, he realized that the best way to make his project work would be to adjust his design decisions so that the existing GNU components would be compatible with his kernel.

The result of Torvalds' work was the release on the net in 1991 of a sketchy working model of a free software kernel for a Unix-like operating system for PCs, fully compatible with and designed convergently with the large and high-quality suite of system components created by Stallman's Project GNU and distributed by the Free Software Foundation. Because Torvalds chose to release the Linux kernel under the Free Software Foundation's General Public License, of which more below, the hundreds and eventually thousands of programmers around the world who chose to contribute their effort towards the further development of the kernel could be sure that their efforts would result in permanently free software that no one could turn into a proprietary product. Everyone knew that everyone else would be able to test, improve, and redistribute their improvements. Torvalds accepted contributions freely, and with a genially effective style maintained overall direction without dampening enthusiasm. The development of the Linux kernel proved that the Internet made it possible to aggregate collections of programmers far larger than any commercial manufacturer could afford, joined almost non-hierarchically in a development project ultimately involving more than one million lines of computer code - a scale of

collaboration among geographically dispersed unpaid volunteers previously unimaginable in human history [20].

By 1994, Linux had reached version 1.0, representing a usable production kernel. Level 2.0 was reached in 1996, and by 1998, with the kernel at 2.2.0 and available not only for x86 machines but for a variety of other machine architectures, GNU/Linux - the combination of the Linux kernel and the much larger body of Project GNU components - and Windows NT were the only two operating systems in the world gaining market share. A Microsoft internal assessment of the situation leaked in October 1998 and subsequently acknowledged by the company as genuine concluded that "Linux represents a best-of-breed UNIX, that is trusted in mission critical applications, and - due to it's [sic] open source code - has a long term credibility which exceeds many other competitive OS's." [21] GNU/Linux systems are now used throughout the world, operating everything from Web servers at major electronic commerce sites to "ad-hoc supercomputer" clusters to the network infrastructure of money-center banks. GNU/Linux is found on the space shuttle, and running behind-the-scenes computers at (yes) Microsoft. Industry evaluations of the comparative reliability of Unix systems have repeatedly shown that Linux is far and away the most stable and reliable Unix kernel, with a reliability exceeded only by the GNU tools themselves. GNU/Linux not only out-performs commercial proprietary Unix versions for PCs in benchmarks, but is renowned for its ability to run, undisturbed and uncomplaining, for months on end in high-volume high-stress environments without crashing.

Other components of the free software movement have been equally successful. Apache, far and away the world's leading Web server program, is free software, as is Perl, the programming language which is the lingua franca for the programmers who build sophisticated Web sites. Netscape Communications now distributes its Netscape Communicator 5.0 browser as free software, under a close variant of the Free Software Foundation's General Public License. Major PC manufacturers, including IBM, have announced plans or are already distributing GNU/Linux as a customer option on their top-of-the-line PCs intended for use as Web- and file servers. Samba, a program that allows GNU/Linux computers to act as Windows NT file servers, is used worldwide as an alternative to Windows NT Server, and provides effective low-end competition to Microsoft in its own home market. By the standards of software quality that have been recognized in the industry for decades - and whose continuing relevance will be clear to you the next time your Windows PC crashes - the news at century's end is unambiguous. The world's most profitable and powerful corporation comes in a distant second, having excluded all but the real victor from the race. Propertarianism joined to capitalist vigor destroyed meaningful commercial competition, but when it came to making good software, anarchism won.

III. Anarchism as a Mode of Production

It's a pretty story, and if only the IPdroid and the econodwarf hadn't been blinded by theory, they'd have seen it coming. But though some of us had been working for it and predicting it for years, the theoretical consequences are so subversive for the thoughtways that maintain our dwarves and droids in comfort that they can hardly be blamed for refusing to see. The facts proved that something was wrong with the "incentives" metaphor that underprops conventional intellectual property reasoning [22]. But they did more. They provided an initial glimpse into the future of human creativity in a world of global interconnection, and it's not a world made for dwarves and droids.

My argument, before we paused for refreshment in the real world, can be summarized this way: Software - whether executable programs, music, visual art, liturgy, weaponry, or what have you - consists of bitstreams, which although essentially indistinguishable are treated by a confusing multiplicity of legal categories. This multiplicity is unstable in the long term for reasons integral to the legal process. The unstable diversity of rules is caused by the need to distinguish among kinds of property interests in bitstreams. This need is primarily felt by those who stand to profit from the socially acceptable forms of monopoly created by treating ideas as property. Those of us who are worried about the social inequity and cultural hegemony created by this intellectually unsatisfying and morally repugnant regime are shouted down. Those doing the shouting, the dwarves and the droids, believe that these property rules are necessary not from any overt yearning for life in Murdochworld - though a little luxurious co-optation is always welcome - but because the metaphor of incentives, which they take to be not just an image but an argument, proves that these rules - despite their lamentable consequences - are necessary if we are to make good software. The only way to continue to believe this is to ignore the facts. At the center of the digital revolution, with the executable bitstreams that make everything else possible, propertarian regimes not only do not make things better, they can make things radically worse. Property concepts, whatever else may be wrong with them, do not enable and have in fact retarded progress.

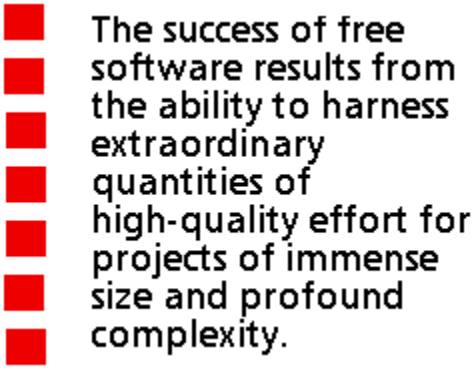
But what is this mysterious alternative? Free software exists, but what are its mechanisms, and how does it generalize towards a non-propertarian theory of the digital society?

The Legal Theory of Free Software

There is a myth, like most myths partially founded on reality, that computer programmers are all libertarians. Right-wing ones are capitalists, cleave to their stock options, and disdain taxes, unions, and civil rights laws; left-wing ones hate the market and all government, believe in strong encryption no matter how much nuclear terrorism it may cause,[23] and dislike Bill Gates because he's rich. There is doubtless a foundation for this belief. But the most significant difference between political thought inside the digirati and outside it is that in the network society, anarchism (or more properly, anti-possessive individualism) is a viable political philosophy.

The center of the free software movement's success, and the greatest achievement of Richard Stallman, is not a piece of computer code. The success of free software, including the overwhelming success of GNU/Linux, results from the ability to harness extraordinary quantities of high-quality effort for projects of immense size and profound complexity. And this

ability in turn results from the legal context in which the labor is mobilized. As a visionary designer Richard Stallman created more than Emacs, GDB, or GNU. He created the General Public License.



The GPL,[24] also known as the copyleft, uses copyright, to paraphrase Toby Milsom, to counterfeit the phenomena of anarchism. As the license preamble expresses it:

> When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.
>
> To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.
>
> For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Many variants of this basic free software idea have been expressed in licenses of various kinds, as I have already indicated. The GPL is different from the other ways of expressing these values in one crucial respect. Section 2 of the license provides in pertinent part:

> You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work ..., provided that you also meet all of these conditions:
>
> ...
>
> b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

Section 2(b) of the GPL is sometimes called "restrictive," but its intention is liberating. It creates a commons, to which anyone may add but from which no one may subtract. Because of §2(b), each contributor to a GPL'd project is assured that she, and all other users, will be able to run, modify and redistribute the program indefinitely, that source code will always be available, and that, unlike commercial software, its longevity cannot be limited by the contingencies of the marketplace or the decisions of future developers. This "inheritance" of the GPL has sometimes been criticized as an example of the free software movement's anti-commercial bias. Nothing could be further from the truth. The effect of §2(b) is to make commercial distributors of free software better competitors against proprietary software businesses. For confirmation of this point, one can do no better than to ask the proprietary competitors. As the author of the Microsoft "Halloween" memorandum, Vinod Vallopillil, put it:

> The GPL and its aversion to code forking reassures customers that they aren't riding an evolutionary `dead-end' by subscribing to a particular commercial version of Linux.
>
> The "evolutionary dead-end" is the core of the software FUD argument [25].

Translated out of Microspeak, this means that the strategy by which the dominant proprietary manufacturer drives customers away from competitors - by sowing fear, uncertainty and doubt about other software's long-term viability - is ineffective with respect to GPL'd programs. Users of GPL'd code, including those who purchase software and systems from a commercial reseller, know that future improvements and repairs will be accessible from the commons, and need not fear either the disappearance of their supplier or that someone will use a particularly attractive improvement or a desperately necessary repair as leverage for "taking the program private."

This use of intellectual property rules to create a commons in cyberspace is the central institutional structure enabling the anarchist triumph. Ensuring free access and enabling modification at each stage in the process means that the evolution of software occurs in the fast Lamarckian mode: each favorable acquired characteristic of others' work can be directly

inherited. Hence the speed with which the Linux kernel, for example, outgrew all of its proprietary predecessors. Because defection is impossible, free riders are welcome, which resolves one of the central puzzles of collective action in a propertarian social system.

Non-propertarian production is also directly responsible for the famous stability and reliability of free software, which arises from what Eric Raymond calls "Linus' law": With enough eyeballs, all bugs are shallow. In practical terms, access to source code means that if I have a problem I can fix it. Because I can fix it, I almost never have to, because someone else has almost always seen it and fixed it first.

For the free software community, commitment to anarchist production may be a moral imperative; as Richard Stallman wrote, it's about freedom, not about price. Or it may be a matter of utility, seeking to produce better software than propertarian modes of work will allow. From the droid point of view, the copyleft represents the perversion of theory, but better than any other proposal over the past decades it resolves the problems of applying copyright to the inextricably merged functional and expressive features of computer programs. That it produces better software than the alternative does not imply that traditional copyright principles should now be prohibited to those who want to own and market inferior software products, or (more charitably) whose products are too narrow in appeal for communal production. But our story should serve as a warning to droids: The world of the future will bear little relation to the world of the past. The rules are now being bent in two directions. The corporate owners of "cultural icons" and other assets who seek ever-longer terms for corporate authors, converting the "limited Time" of Article I, §8 into a freehold have naturally been whistling music to the android ear [26]. After all, who bought the droids their concert tickets? But as the propertarian position seeks to embed itself ever more strongly, in a conception of copyright liberated from the minor annoyances of limited terms and fair use, at the very center of our "cultural software" system, the anarchist counter-strike has begun. Worse is yet to befall the droids, as we shall see. But first, we must pay our final devoirs to the dwarves.

Because It's There: Faraday's Magnet and Human Creativity

After all, they deserve an answer. Why do people make free software if they don't get to profit? Two answers have usually been given. One is half-right and the other is wrong, but both are insufficiently simple.

The wrong answer is embedded in numerous references to "the hacker gift-exchange culture." This use of ethnographic jargon wandered into the field some years ago and became rapidly, if misleadingly, ubiquitous. It reminds us only that the econometricians have so corrupted our thought processes that any form of non-market economic behavior seems equal to every other kind. But gift-exchange, like market barter, is a propertarian institution. Reciprocity is central to these symbolic enactments of mutual dependence, and if either the yams or the fish are short-weighted, trouble results. Free software, at the risk of repetition, is a commons: no reciprocity ritual is enacted there. A few people give away code that others sell, use, change, or borrow wholesale to lift out parts for something else. Notwithstanding the very large number of people (tens of thousands, at most) who have contributed to GNU/Linux, this is orders of magnitude less than the number of users who make no contribution whatever [27].

A part of the right answer is suggested by the claim that free software is made by those who seek reputational compensation for their activity. Famous Linux hackers, the theory is, are known all over the planet as programming deities. From this they derive either enhanced self-esteem or indirect material advancement [28]. But the programming deities, much as they have contributed to free software, have not done the bulk of the work. Reputations, as Linus Torvalds himself has often pointed out, are made by willingly acknowledging that it was all done by someone else. And, as many observers have noted, the free software movement has also produced superlative documentation. Documentation-writing is not what hackers do to attain cool, and much of the documentation has been written by people who didn't write the code. Nor must we limit the indirect material advantages of authorship to increases in reputational capital. Most free software authors I know have day jobs in the technology industries, and the skills they hone in the more creative work they do outside the market no doubt sometimes measurably enhance their value within it. And as the free software products gained critical mass and became the basis of a whole new set of business models built around commercial distribution of that which people can also get for nothing, an increasing number of people are specifically employed to write free software. But in order to be employable in the field, they must already have established themselves there. Plainly, then, this motive is present, but it isn't the whole explanation.

Indeed, the rest of the answer is just too simple to have received its due. The best way to understand is to follow the brief and otherwise unsung career of an initially-grudging free software author. Microsoft's Vinod Vallopillil, in the course of writing the competitive analysis of Linux that was leaked as the second of the famous "Halloween memoranda," bought and installed a Linux system on one of his office computers. He had trouble because the (commercial) Linux distribution he installed did not contain a daemon to handle the DHCP protocol for assignment of dynamic IP addresses. The result was important enough for us to risk another prolonged exposure to the Microsoft Writing Style:

> A small number of Web sites and FAQs later, I found an FTP site with a Linux DHCP client. The DHCP client was developed by an engineer employed by Fore Systems (as evidenced by his e-mail address; I believe, however, that it was developed in his own free time). A second set of documentation/manuals was written for the DHCP client by a hacker in *Hungary* which provided relatively simple instructions on how to install/load the client.
>
> I downloaded & uncompressed the client and typed two simple commands:
>
> Make - compiles the client binaries

> Make Install -installed the binaries as a Linux Daemon
>
> Typing "DHCPCD" (for DHCP Client Daemon) on the command line triggered the DHCP discovery process and voila, I had IP networking running.
>
> Since I had just downloaded the DHCP client code, on an impulse I played around a bit. Although the client wasn't as extensible as the DHCP client we are shipping in NT5 (for example, it won't query for arbitrary options & store results), it was obvious how I could write the additional code to implement this functionality. The full client consisted of about 2,600 lines of code.
>
> One example of esoteric, extended functionality that was clearly patched in by a third party was a set of routines to that would pad the DHCP request with host-specific strings required by Cable Modem / ADSL sites.
>
> A few other steps were required to configure the DHCP client to auto-start and auto-configure my Ethernet interface on boot but these were documented in the client code and in the DHCP documentation from the Hungarian developer.
>
> I'm a poorly skilled UNIX programmer but it was immediately obvious to me how to incrementally extend the DHCP client code (the feeling was exhilarating and addictive).
>
> Additionally, due directly to GPL + having the full development environment in front of me, I was in a position where I could write up my changes and e-mail them out within a couple of hours (in contrast to how things like this would get done in NT). Engaging in that process would have prepared me for a larger, more ambitious Linux project in the future [29].

"The feeling was exhilarating and addictive." Stop the presses: Microsoft experimentally verifies Moglen's Metaphorical Corollary to Faraday's Law. Wrap the Internet around every brain on the planet and spin the planet. Software flows in the wires. It's an emergent property of human minds to create. "Due directly to the GPL," as Vallopillil rightly pointed out, free software made available to him an exhilarating increase in his own creativity, of a kind not achievable in his day job working for the Greatest Programming Company on Earth. If only he had e-mailed that first addictive fix, who knows where he'd be now?

So, in the end, my dwarvish friends, it's just a human thing. Rather like why Figaro sings, why Mozart wrote the music for him to sing to, and why we all make up new words: Because we can. Homo ludens, meet Homo faber. The social condition of global interconnection that we call the Internet makes it possible for all of us to be creative in new and previously undreamed-of ways. Unless we allow "ownership" to interfere. Repeat after me, ye dwarves and men: Resist the resistance!

IV. Their Lordships Die in the Dark?

For the IPdroid, fresh off the plane from a week at Bellagio paid for by Dreamworks SKG, it's enough to cause indigestion.

Unlock the possibilities of human creativity by connecting everyone to everyone else? Get the ownership system out of the way so that we can all add our voices to the choir, even if that means pasting our singing on top of the Mormon Tabernacle and sending the output to a friend? No one sitting slack-jawed in front of a televised mixture of violence and imminent copulation carefully devised to heighten the young male eyeball's interest in a beer commercial? What will become of civilization? Or at least of copyright teachers?

But perhaps this is premature. I've only been talking about software. Real software, the old kind, that runs computers. Not like the software that runs DVD players, or the kind made by the Grateful Dead. "Oh yes, the Grateful Dead. Something strange about them, wasn't there? Didn't prohibit recording at their concerts. Didn't mind if their fans rather riled the recording industry. Seem to have done all right, though, you gotta admit. Senator Patrick Leahy, isn't he a former Deadhead? I wonder if he'll vote to extend corporate authorship terms to 125 years, so that Disney doesn't lose The Mouse in 2004. And those DVD players - they're computers, aren't they?"

In the digital society, it's all connected. We can't depend for the long run on distinguishing one bitstream from another in order to figure out which rules apply. What happened to software is already happening to music. Their recording industry lordships are now scrambling wildly to retain control over distribution, as both musicians and listeners realize that the middlepeople are no longer necessary. The Great Potemkin Village of 1999, the so-called Secure Digital Music Initiative, will have collapsed long before the first Internet President gets inaugurated, for simple technical reasons as obvious to those who know as the ones that dictated the triumph of free software [30]. The anarchist revolution in music is different from the one in software *tout court*, but here too - as any teenager with an MP3 collection of self-released music from unsigned artists can tell you - theory has been killed off by the facts. Whether you are Mick Jagger, or a great national artist from the third world looking for a global audience, or a garret-dweller reinventing music, the recording industry will soon have nothing to offer you that you can't get better for free. And music doesn't sound worse when distributed for free, pay what you want directly to the artist, and don't pay anything if you don't want to. Give it to your friends; they might like it.

What happened to music is also happening to news. The wire services, as any U.S. law student learns even before taking the near-obligatory course in Copyright for Droids, have a protectible property interest in their expression of the news, even if not in the facts the news reports [31]. So why are they now giving all their output away? Because in the world of

the Net, most news is commodity news. And the original advantage of the news gatherers, that they were internally connected in ways others were not when communications were expensive, is gone. Now what matters is collecting eyeballs to deliver to advertisers. It isn't the wire services that have the advantage in covering Kosovo, that's for sure. Much less those paragons of "intellectual" property, their television lordships. They, with their overpaid pretty people and their massive technical infrastructure, are about the only organizations in the world that can't afford to be everywhere all the time. And then they have to limit themselves to ninety seconds a story, or the eyeball hunters will go somewhere else. So who makes better news, the propertarians or the anarchists? We shall soon see.

Oscar Wilde says somewhere that the problem with socialism is that it takes up too many evenings. The problems with anarchism as a social system are also about transaction costs. But the digital revolution alters two aspects of political economy that have been otherwise invariant throughout human history. All software has zero marginal cost in the world of the Net, while the costs of social coordination have been so far reduced as to permit the rapid formation and dissolution of large-scale and highly diverse social groupings entirely without geographic limitation [32]. Such fundamental change in the material circumstances of life necessarily produces equally fundamental changes in culture. Think not? Tell it to the Iroquois. And of course such profound shifts in culture are threats to existing power relations. Think not? Ask the Chinese Communist Party. Or wait 25 years and see if you can find them for purposes of making the inquiry.

In this context, the obsolescence of the IPdroid is neither unforseeable nor tragic. Indeed it may find itself clanking off into the desert, still lucidly explaining to an imaginary room the profitably complicated rules for a world that no longer exists. But at least it will have familiar company, recognizable from all those glittering parties in Davos, Hollywood, and Brussels. Our Media Lords are now at handigrips with fate, however much they may feel that the Force is with them. The rules about bitstreams are now of dubious utility for maintaining power by co-opting human creativity. Seen clearly in the light of fact, these Emperors have even fewer clothes than the models they use to grab our eyeballs. Unless supported by user-disabling technology, a culture of pervasive surveillance that permits every reader of every "property" to be logged and charged, and a smokescreen of droid-breath assuring each and every young person that human creativity would vanish without the benevolent aristocracy of BillG the Creator, Lord Murdoch of Everywhere, the Spielmeister and the Lord High Mouse, their reign is nearly done. But what's at stake is the control of the scarcest resource of all: our attention. Conscripting that makes all the money in the world in the digital economy, and the current lords of the earth will fight for it. Leagued against them are only the anarchists: nobodies, hippies, hobbyists, lovers, and artists. The resulting unequal contest is the great political and legal issue of our time. Aristocracy looks hard to beat, but that's how it looked in 1788 and 1913 too. It is, as Chou En-Lai said about the meaning of the French Revolution, too soon to tell.

About the Author

Eben Moglen is Professor of Law & Legal History, Columbia Law School.
E-mail: Mail: moglen@columbia.edu

Acknowledgments

This paper was prepared for delivery at the Buchmann International Conference on Law, Technology and Information, at Tel Aviv University, May 1999; my thanks to the organizers for their kind invitation. I owe much as always to Pamela Karlan for her insight and encouragement. I especially wish to thank the programmers throughout the world who made free software possible.

Notes

1. The distinction was only approximate in its original context. By the late 1960's certain portions of the basic operation of hardware were controlled by programs digitally encoded in the electronics of computer equipment, not subject to change after the units left the factory. Such symbolic but unmodifiable components were known in the trade as "microcode," but it became conventional to refer to them as "firmware." Softness, the term "firmware" demonstrated, referred primarily to users' ability to alter symbols determining machine behavior. As the digital revolution has resulted in the widespread use of computers by technical incompetents, most traditional software - application programs, operating systems, numerical control instructions, and so fort - is, for most of its users, firmware. It may be symbolic rather than electronic in its construction, but they couldn't change it even if they wanted to, which they often - impotently and resentfully - do. This "firming of software" is a primary condition of the propertarian approach to the legal organization of digital society, which is the subject of this paper.

2. Within the present generation, the very conception of social "development" is shifting away from possession of heavy industry based on the internal-combustion engine to "post-industry" based on digital communications and the related "knowledge-based" forms of economic activity.

3. Actually, a moment's thought will reveal, our genes are firmware. Evolution made the transition from analog to digital before the fossil record begins. But we haven't possessed the power of controlled direct modification. Until the day before yesterday. In the next century the genes too will become software, and while I don't discuss the issue further in this paper, the political consequences of unfreedom of software in this context are even more disturbing than they are with respect to cultural artifacts.

4. *See, e.g.,* J. M. Balkin, 1998. *Cultural Software: a Theory of Ideology.* New Haven: Yale University Press.

5. *See* Henry Sumner Maine, 1861. *Ancient Law: Its Connection with the Early History of Society, and Its Relation to Modern Idea.* First edition. London: J. Murray.

6. In general I dislike the intrusion of autobiography into scholarship. But because it is here my sad duty and great pleasure to challenge the qualifications or *bona fides* of just about everyone, I must enable the assessment of my own. I was first exposed to the craft of computer programming in 1971. I began earning wages as a commercial programmer in 1973 - at the age of thirteen - and did so, in a variety of computer services, engineering, and multinational technology enterprises, until 1985. In 1975 I helped write one of the first networked e-mail systems in the United States; from 1979 I was engaged in research and development of advanced computer programming languages at IBM. These activities made it economically possible for me to study the arts of historical scholarship and legal cunning. My wages were sufficient to pay my tuitions, but not - to anticipate an argument that will be made by the econodwarves further along - because my programs were the intellectual property of my employer, but rather because they made the hardware my employer sold work better. Most of what I wrote was effectively free software, as we shall see. Although I subsequently made some inconsiderable technical contributions to the actual free software movement this paper describes, my primary activities on its behalf have been legal: I have served for the past five years (without pay, naturally) as general counsel of the Free Software Foundation.

7. The player, of course, has secondary inputs and outputs in control channels: buttons or infrared remote control are input, and time and track display are output.

8. This is not an insight unique to our present enterprise. A closely-related idea forms one of the most important principles in the history of Anglo-American law, perfectly put by Toby Milsom in the following terms:

> The life of the common law has been in the abuse of its elementary ideas. If the rules of property give what now seems an unjust answer, try obligation; and equity has proved that from the materials of obligation you can counterfeit the phenomena of property. If the rules of contract give what now seems an unjust answer, try tort. ... If the rules of one tort, say deceit, give what now seems an unjust answer, try another, try negligence. And so the legal world goes round.

S.F.C. Milsom, 1981. *Historical Foundations of the Common Law.* Second edition. London: Butterworths, p. 6.

9. *See* Isaiah Berlin, 1953. *The Hedgehog and the Fox: An Essay on Tolstoy's View of History.* New York: Simon and Schuster.

10. *See* The Virtual Scholar and Network Liberation.

11. Some basic vocabulary is essential. Digital computers actually execute numerical instructions: bitstrings that contain information in the "native" language created by the machine's designers. This is usually referred to as "machine language." The machine languages of hardware are designed for speed of execution at the hardware level, and are not suitable for direct use by human beings. So among the central components of a computer system are "programming languages," which translate expressions convenient for humans into machine language. The most common and relevant, but by no means the only, form of computer language is a "compiler." The compiler performs static translation, so that a file containing human-readable instructions, known as "source code" results in the generation of one or more files of executable machine language, known as "object code."

12. This, I should say, was the path that most of my research and development followed, largely in connection with a language called APL ("A Programming Language") and its successors. It was not, however, the ultimately-dominant approach, for reasons that will be suggested below.

13. This description elides some details. By the mid-1970's IBM had acquired meaningful competition in the mainframe computer business, while the large-scale antitrust action brought against it by the U.S. government prompted the decision to "unbundle," or charge separately, for software. In this less important sense, software ceased to be free. But - without entering into the now-dead but once-heated controversy over IBM's software pricing policies - the unbundling revolution had less effect on the social practices of software manufacture than might be supposed. As a fellow responsible for technical improvement of one programming language product at IBM from 1979 to 1984, for example, I was able to treat the product as "almost free," that is, to discuss with users the changes they had proposed or made in the programs, and to engage with them in cooperative development of the product for the benefit of all users.

14. This description is highly compressed, and will seem both overly simplified and unduly rosy to those who also worked in the industry during this period of its development. Copyright protection of computer software was a controversial subject in the 1970's, leading to the famous CONTU commission and its mildly pro-copyright recommendations of 1979. And IBM seemed far less cooperative to its users at the time than this sketch makes out. But the most important element is the contrast with the world created by the PC, the Internet, and the dominance of Microsoft, with the resulting impetus for the free software movement, and I am here concentrating on the features that express that contrast.

15. I discuss the importance of PC software in this context, the evolution of "the market for eyeballs" and "the sponsored life" in other chapters of my forthcoming book, *The Invisible Barbecue*, of which this essay forms a part.

16. This same pattern of ambivalence, in which bad programming leading to widespread instability in the new technology is simultaneously frightening and reassuring to technical incompetents, can be seen also in the primarily-American phenomenon of Y2K hysteria.

17. The critical implications of this simple observation about our metaphors are worked out in "How Not to Think about 'The Internet'," in *The Invisible Barbecue*, forthcoming.

18. Technical readers will again observe that this compresses developments occurring from 1969 through 1973.

19. Operating systems, even Windows (which hides the fact from its users as thoroughly as possible), are actually collections of components, rather than undivided unities. Most of what an operating system does (manage file systems, control process execution, etc.) can be abstracted from the actual details of the computer hardware on which the operating system runs. Only a small inner core of the system must actually deal with the eccentric peculiarities of particular hardware. Once the operating system is written in a general language such as C, only that inner core, known in the trade as the kernel, will be highly specific to a particular computer architecture.

20. A careful and creative analysis of how Torvalds made this process work, and what it implies for the social practices of creating software, was provided by Eric S. Raymond in his seminal 1997 paper, The Cathedral and the Bazaar, which itself played a significant role in the expansion of the free software idea.

21. This is a quotation from what is known in the trade as the "Halloween memo," which can be found, as annotated by Eric Raymond, to whom it was leaked, at http://www.opensource.org/halloween/halloween1.html.

22. As recently as early 1994 a talented and technically competent (though Windows-using) law and economics scholar at a major U.S. law school confidently informed me that free software couldn't possibly exist, because no one would have any incentive to make really sophisticated programs requiring substantial investment of effort only to give them away.

23. This question too deserves special scrutiny, encrusted as it is with special pleading on the state-power side. See my brief essay "*So Much for Savages*: Navajo 1, Government 0 in Final Moments of Play."

24. *See* GNU General Public License, Version 2, June 1991.

25. V. Vallopillil, Open Source Software: A (New?) Development Methodology.

26. The looming expiration of Mickey Mouse's ownership by Disney requires, from the point of view of that wealthy "campaign contributor," for example, an alteration of the general copyright law of the United States. See "Not Making it Any More? Vaporizing the Public Domain," in *The Invisible Barbecue*, forthcoming.

27. A recent industry estimate puts the number of Linux systems worldwide at 7.5 million. *See* Josh McHugh, 1998. "Linux: The Making of a Global Hack," *Forbes* (August 10). Because the software is freely obtainable throughout the Net, there is no simple way to assess actual usage.

28. Eric Raymond is a partisan of the "ego boost" theory, to which he adds another faux-ethnographic comparison, of free software composition to the Kwakiutl potlatch. *See* Eric S. Raymond, 1998. Homesteading the Noosphere.. But the potlatch, certainly a form of status competition, is unlike free software for two fundamental reasons: it is essentially hierarchical, which free software is not, and, as we have known since Thorstein Veblen first called attention to its significance, it is a form of conspicuous waste. *See* Thorstein Veblen, 1967. *The Theory of the Leisure Class.* New York: Viking, p. 75. These are precisely the grounds which distinguish the anti-hierarchical and utilitiarian free software culture from its propertarian counterparts.

29. Vinod Vallopillil, Linux OS Competitive Analysis (Halloween II). Note Vallopillil's surprise that a program written in California had been subsequently documented by a programmer in Hungary.

30. See "They're Playing Our Song: The Day the Music Industry Died," in *The Invisible Barbecue*, forthcoming.

31. International News Service v. Associated Press, 248 U.S. 215 (1918). With regard to the actual terse, purely functional expressions of breaking news actually at stake in the jostling among wire services, this was always a distinction only a droid could love.

32. See "No Prodigal Son: The Political Theory of Universal Interconnection," in *The Invisible Barbecue*, forthcoming.

Contents Index