Anarquismo triunfante: el software libre y la muerte del copyright

Eben Moglen

La propagación del kernel del sistema operativo Linux ha llamado la atención hacia el movimiento del Software Libre. El presente ensayo muestra por qué el Software Libre, lejos de ser un participante marginal en el Mercado del Software comercial, es el vital primer paso hacia el marchitamiento del sistema de propiedad intelectual.

Universidad Libre de Santiago
Ediciones RedHack.Lab / 2010
Pirateado desde: El Infinito Perpendicular
Original en: BlagBlagBlag

Contenidos

Eben Moglen es un profesor de derecho e historia en la Universidad de Columbia. Es director del Software Freedom Law Center, que fundó en 2005. Y colabora como consejero general para la Fundación del Software Libre. Wikipedia

I. El software como propiedad: la paradoja teórica

SOFTWARE: ninguna otra palabra connota tan exhaustivamente tanto los efectos prácticos como los sociales de la revolución digital. En sus inicios, el término era exclusivamente técnico, y denotaba las partes de algún sistema de cómputo que, en contraste con el hardware (el cual era manufacturado en sistemas electrónicos sin la posibilidad de realizarle ajustes posteriores al gusto y necesidad del usuario), podía ser cambiado libremente. El primer software se refería únicamente a la configuración y conexión de cables o interruptores en los paneles exteriores de un dispositivo electrónico, pero tan pronto como se desarrollaron los medios lingüísticos para alterar el funcionamiento y el comportamiento de los equipos de cómputo, la palabra "software" comenzó a denotar sobre todo las expresiones en un lenguaje más o menos inteligible para los seres humanos, que describía y controlaba el funcionamiento y comportamiento de las máquinas [1].

1. La distinción era sólo aproximada en su contexto original. A partir de los últimos años de la década de 1960, ciertas porciones de la operación básica del hardware eran controladas por programas codificados digitalmente en los dispositivos electrónicos del equipo de cómputo, y no estaban sujetas a ser modificadas después de que las unidades salieran de la fábrica. Estos componentes simbólicos e inmodificables eran conocidos en el rubro como microcódigo, pero después se tornó convencional referirse a ellos como firmware. El término firmware demostró que la parte suave (soft, en referencia al software), se refería principalmente a la capacidad que tenían los usuarios para alterar los símbolos que determinaban el comportamiento de la máquina. Puesto que la revolución digital ha resultado en un uso amplísimo por personas técnicamente incompetentes, una gran parte del software tradicional (programas de aplicaciones, sistemas operativos, instrucciones de control numérico, y demás) es, para la mayoría de sus usuarios, firmware. Puede que sea simbólico en vez de electrónico en su construcción, pero no podrían cambiarlo aunque quisieran, cosa que, con impotencia y resentimiento, les sucede a menudo. Este "endurecimiento" del software es una condición principal del enfoque propietario de la organización legal de la sociedad digital, que es el tema de este documento.

Así fue entonces y así es ahora. La tecnología basada en la manipulación de información codificada digitalmente es ahora socialmente dominante en la mayoría de los aspectos de la cultura humana en las sociedades "desarrolladas" [2]. El cambio de la representación análoga por la digital (en video, audio, impresos, telecomunicaciones e incluso coreografías, manifestaciones religiosas y estímulos sexuales) convierte potencialmente a todas las formas simbólicas de actividad humana en software, es decir, en instrucciones modificables para describir y controlar el desempeño de las máquinas. Por una retro-formación semántica característica del pensamiento cientificista occidental, la división entre hardware y software se observa ahora en el mundo natural o social, y se ha vuelto una nueva forma de expresar el conflicto entre ideas de determinismo y libre albedrío, naturaleza y crianza, o genes y cultura. Nuestro "hardware", cableado genéticamente, es nuestra naturaleza, y nos determina. Nuestra crianza es "software", establece nuestra programación cultural, la cual es nuestra libertad comparativa. Y demás comparaciones, para aquellos incautos ante las tonterías. [3] Así pues el "software se convierte en una metáfora viable para toda la actividad simbólica, aparentemente divorciada del contexto técnico del origen de la palabra, a

pesar de la incomodidad despertada en los técnicamente competentes cuando el término es usado de ese modo, omitiendo el significado conceptual de su derivación [4].

2. En la generación actual, el concepto mismo de “desarrollo” social se está alejando de la posesión de industria pesada, basada en motores de combustión interna, hacia una “post-industria” basada en comunicaciones digitales, y las formas de actividad económica “basadas en conocimiento”.

3. De hecho, reflexionar un momento revelará que nuestros genes son firmware. La evolución hizo la transición de análogo a digital antes del inicio de los registros fósiles. Pero no nos hemos adueñado del poder de hacer modificaciones directas controladas, hasta anteayer. En el próximo siglo, también los genes se convertirán en software, y aunque no discuto más el asunto en este ensayo, las consecuencias políticas de la no libertad del software en tal contexto son incluso más preocupantes de lo que son respecto de los artefactos culturales.

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

Pero la adopción generalizada de la tecnología digital para uso de aquellos que no entienden los principios de su operación, mientras que aparentemente consiente el amplio empleo metafórico del término "software", no nos permite de hecho ignorar a las computadoras que están ahora en todos lados por debajo de nuestro tejido social. El movimiento de lo analógico a lo digital es más importante para la estructura de las relaciones sociales y legales que el más famoso aunque menos seguro movimiento del estatus al contrato [5]. Estas son malas noticias para aquellos pensadores legales que no lo entienden, razón por la cual tantas pretensiones de entender siguen tan floridamente aconteciendo. Potencialmente, sin embargo, nuestra gran transición representa muy buenas noticias para aquellos que pueden convertir esta tierra recién descubierta en propiedad para ellos mismos. Razón por la cual los "dueños" contemporáneos del software impulsan y alientan con tanta fuerza la ignorancia de todos los demás. Desafortunadamente para ellos - por razones familiares para los teóricos legales que no han entendido aún como aplicar su lógica tradicional en este campo - el truco no funcionará. Este ensayo explica por qué [6].

5. Ver 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. En general me disgustan las irrupciones de las autobiografías en las publicaciones académicas. Pero puesto que tengo aquí el triste deber y el enorme placer de desafiar las calificaciones o la buena fe de casi todo el mundo, debo permitirme una evaluación propia. Estuve expuesto por primera vez al arte de la programación de computadoras en 1971. Empecé a recibir ingresos como programador comercial en 1973 -a la edad de trece- y lo hice, en una variedad de servicios de cómputo, ingenierías computacionales, y empresas multinacionales de tecnología hasta 1985. En 1975 ayudé a escribir uno de los primeros sistemas de e-mail de los Estados Unidos; a partir de 1979 estuve involucrado en investigación y desarrollo de lenguajes avanzados de programación de computadoras en IBM. Estas actividades hicieron posible económicamente para mí el estudiar las artes de la erudición histórica y de los artificios legales. Mis ingresos eran suficientes para

pagar mis clases, pero no - para anticiparme a un argumento que harán más adelante los enanos economistas - porque mis programas fueran la propiedad intelectual de mi empleador, sino porque hacían que el hardware que mi empleador vendía funcionara mejor. La mayor parte de lo que escribí fue efectivamente software libre, como veremos. A pesar de que subsecuentemente hice algunas contribuciones técnicas insignificantes al movimiento por el software libre que este ensayo describe, mis actividades principales en su beneficio ha sido legales: He servido por los pasados cinco años (sin un salario, naturalmente) como asesor jurídico general de la Fundación para el Software Libre.

Necesitamos empezar por considerar la esencia técnica de los instrumentos familiares que nos rodean en la era el "software cultural". Un reproductor de CDs es un buen ejemplo. Su input primario es una cadena de bits leída desde un disco de almacenamiento óptico. La cadena de bits describe la música en términos de medidas de frecuencia y amplitud tomadas 44,000 veces por segundo en cualquiera de los dos canales de audio. El output primario del reproductor son señales de audio análogas [7]. Como todo lo demás en el mundo digital, la música vista por un reproductor de CDs es mera información numérica; una grabación particular de la Novena sinfonía de Beethoven dirigida por Arturo Toscanini con la Orquesta Sinfónica y coral de la NBC es (por decir un dígito insignificante) 1276749873424, mientras que la última grabación particularmente perversa de Glenn Gould dirigiendo las Variaciones Goldberg es (similarmente aunque truncado) 767459083268.

De manera bastante extraña, esos dos números tienen su "registro copyright". Esto quiere decir, supuestamente, que no puedes poseer otra copia de estos números, una vez fijados en cualquier forma física, a menos que tengas la licencia. Y no puedes convertir a 767459083268 en 2347895697 para tus amigos (corrigiendo así el juicio ridículo de Gould respecto de los tempi) sin hacer un "trabajo derivado", para el cual es necesaria una licencia.

Al mismo tiempo, otro disco de almacenamiento óptico similar contiene otro número, digamos el 7537489532. Este es un algoritmo para programación lineal de grandes sistemas con restricciones múltiples, útil por ejemplo si quieres hacer un uso óptimo de tu material rodante al gestionar un ferrocarril de carga. Dicho número (en los E.U.) está "patentado", lo que quiere decir que no puedes deducir el 7537489532 para ti mismo, o de otro modo "practicar el arte" de lo patentado respecto a resolver problemas de programación lineal sin importar cómo llegues a la idea, incluyendo el haberla descubierto tú mismo, a no ser que el dueño de ese número te haya otorgado una licencia.

7. El reproductor, por supuesto, tiene inputs y outputs secundarios en los canales de control: los botones o el control remoto infrarrojo son inputs, y el tiempo y número de pista en la pantalla son outputs.

Entonces está también el 9892454959483. Este es el código fuente de Microsoft Word. El cual además de tener "copyright" es un secreto comercial. Eso quiere decir que si tomas ese número de Microsoft y se lo das a cualquier otra persona puedes ser castigado.

Finalmente, está el 588832161316. No hace nada, sólo es el cuadrado de 767354. Hasta

donde sé, no es propiedad de nadie bajo ninguna de estas categorías. Aún.

En este punto debemos enfrentar nuestra primera objeción respecto de lo recién aprendido. Viene de una criatura conocida como el androidePI. Dicho androide tiene una mente sofisticada y una vida cultivada. Aprecia muchísimo las cenas elegantes en conferencias académicas y ministeriales acerca de los ADPICs, sin mencionar el privilegio de las apariciones frecuentes en MSNBC. Y quiere que sepas que estoy cometiendo un error al confundir la representación con la propiedad intelectual en sí misma. No es el número lo que fue patentado, estúpido, sólo el algoritmo Karmarkar. El número puede tener copyright, porque el copyright cubre las cualidades expresivas de la representación particular tangible de una idea (en la cual algunas propiedades funcionales podrían fusionarse misteriosamente, siempre y cuando no estén demasiado fusionadas), pero no el algoritmo. Mientras que el número no puede patentarse, sólo la "enseñanza" del número con respecto a hacer que los ferrocarriles funcionen a tiempo. Y el número que representa el código fuente de Microsoft Word puede ser un secreto comercial, pero si logras deducirlo tú mismo (efectuando manipulaciones aritméticas de otros números publicados por Microsoft, por ejemplo, lo cual es conocido como "ingeniería inversa"), no vas a ser castigado, al menos si vives en algunas partes de los Estados Unidos.

Este androide, como otros androides, frecuentemente está en lo cierto. La condición de ser un androide es conocer todo acerca de algo y nada de todo lo demás. Debido a su oportuna y urgente intervención el androide estableció que el sistema de propiedad intelectual actual contiene muchos aspectos intrincados e ingeniosos. Las complejidades se combinan para permitirle a los profesores ser eruditos, a los políticos obtener contribuciones para sus campañas, a los abogados vestir lindos trajes y mocasines con borlas, y a Murdoch ser rico. Dichas complejidades evolucionaron en su mayor parte en una era de distribución industrial de la información, cuando la información estaba grabada en formas análogas sobre objetos físicos que costaban algo significativo al hacerlos, desplazarlos, y venderlos. Cuando se aplican a la información digital que se mueve sin fricción a través de la red y tiene un costo marginal por copia de cero, todo sigue funcionando, en su mayor parte, mientras no dejes de hacerte de la vista gorda.

Pero yo no estaba argumentando acerca de eso. Quería señalar algo más: que nuestro mundo consiste casa vez más tan sólo de grandes números (también conocidos como cadenas de bits), y que - por razones que no tienen nada que ver con las propiedades emergentes de los números en sí mismas - el sistema legal está actualmente comprometido en tratar a números similares de modo radicalmente diferente. Nadie puede decir, al mirar un número de 100 millones de dígitos de longitud, si ese número está patentado, si tiene copyright, o si es un secreto comercial, o si de hecho es propiedad de alguien. Así que el sistema legal que tenemos - bendecidos como lo somos por sus consecuencias si somos profesores de copyright, políticos, abogados de empresas o el gran Rupert mismo - está obligado a tratar cosas indistinguibles de modos distintos.

Ahora bien, en mi rol como historiador jurídico ocupado del desarrollo secular (esto es, a muy largo plazo) del pensamiento legal, afirmo que los regímenes legales basados en distinciones tajantes pero impredecibles entre objetos similares son radicalmente inestables. Se colapsan con el tiempo porque cada instancia de la aplicación de las

reglas es una invitación para que al menos una parte pueda declarar que en lugar de corresponderle a la categoría ideal A el objeto particular en disputa en cambio debería considerarselo como correspondiente a la categoría B, donde las reglas serán más favorables al partido que hace esa afirmación. Este juego --sobre si una máquina de escribir debe ser considerada un instrumento musical para propósitos de regulación de tarifas de ferrocarriles, o si una excavadora es un vehículo motorizado-- es la materia frecuente de la ingenuidad legal. Pero cuando las categorías legales convencionalmente aprobadas requieren de jueces para distinguir entre lo idéntico, el juego se vuelve infinitamente largo, infinitamente costoso, y casi infinitamente ofensivo para el espectador imparcial [8].

Por lo tanto las parte pueden gastar todo el dinero que quieran en todos los legisladores y jueces que puedan comprar -- que para los nuevos "dueños" del mundo digital son bastantes-- pero las reglas que compren no van a funcionar al final. Tarde o temprano, los paradigmas se van a colapsar. Claro, si "tarde" quiere decir dentro de dos generaciones, la distribución de riqueza y poder santificada en el transcurso podría no ser reversible por ninguna vía menos drástica que una bellum servile de televidentes aplatanados contra magnates de medios masivos. Así que no es suficiente saber que la historia no está del lado de Bill Gates. Estamos prediciendo el futuro en un sentido muy limitado: sabemos que las reglas existentes, las cuales todavía tienen el fervor de las creencias convencionales respaldándolas sólidamente, ya no tienen sentido. Las distintas partes usarán y abusarán de dichas reglas libremente hasta que la mayoría de la opinión conservadora "respetable" reconozca su muerte, con resultados inciertos. Pero la academia realista ya debería estar volteando su atención hacia la clara necesidad de nuevos modos de pensar.

8. Este no es un enfoque único de nuestro presente esfuerzo. Una idea relacionada muy de cerca forma uno de los principios más importantes en la historia legal Anglo-Americana, es expresada a la perfección por Toby Milson en los siguientes términos:

La ley común ha vivido en el abuso de sus ideas elementales. Si las reglas de la propiedad nos dan lo que hoy parece una respuesta injusta, hay que intentar con la obligación; y la equidad ha demostrado que desde los materiales de la obligación puedes falsificar el fenómeno de la propiedad. Si las reglas contractuales nos dan lo que parece una respuesta injusta, hay que intentar el agravio. ... Si las reglas de un agravio, digamos el engaño, nos dan lo que parece una respuesta injusta, intenta otra, intenta la negligencia. Y así se las gasta el mundo legal.

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

Cuando llegamos a este punto de la discusión, nos encontramos en contienda con el otro protagonista principal de la idiotez educada: el econo-enano. Igual que el androideIP, el econo-enano es una especie de erizo[9], pero ahí donde el androide está comprometido con la lógica por experiencia, el econo-enano se especializa en en una visión de la naturaleza humana enérgica y bien enfocada pero totalmente errada. De acuerdo con la visión del econo-enano, cada ser humano es un individuo en posesión de "incentivos", los cuales pueden ser desenterrados retrospectivamente imaginando el estado de su cuenta de banco en varios momentos. Así que en esta instancia el econo-

enano se siente obligado a objetar que sin las reglas sobre las que estoy satirizando, no habría incentivos para crear las cosas que las reglas tratan como propiedad privada: sin la habilidad de excluir a otros de la música no habría música, porque nadie podría estar seguro de lograr que le pagaran por crearla.

La música no es realmente nuestro tema; el software que estoy considerando en este momento es del viejo tipo: programas de computadora. Pero como el econo-enano está determinado a lidiar al menos superficialmente con el tema, y puesto que, como ya vimos, ya no es realmente posible distinguir programas computacionales de ejecuciones musicales, debemos decir una o dos palabras al respecto. Al menos podemos tener la satisfacción de permitirnos un argumento ad pygmeam. Cuando el econo-enano se vuelve rico, según mi experiencia, va a la ópera. Pero sin importar cuan frecuentemente escuche el Don Giovanni nunca se le ocurre que el destino de Mozart debería, según su lógica, haber disuadido a Beethoven, o que tenemos La flauta mágica incluso a pesar de que Mozart sabía muy bien que no le pagarían por ella. De hecho, La flauta mágica, la Pasión según San Mateo, y los motetes del asesino de su esposa Carlo Gesualdo son todas parte de la tradición de siglos de antigüedad del software libre, en el sentido más general, que el econo-enano nunca reconoce del todo.

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

El problema básico del enano es que "los incentivos" son meramente una metáfora, y como metáfora para describir la actividad creativa humana es bastante mala. Lo he dicho antes[10], pero la mejor metáfora surgió el día en que Michael Faraday notó por primera vez qué pasaba cuando enrolló un cable de cobre al rededor de un magneto y le dio vueltas. La corriente fluye en dicho cable, pero no nos preguntamos cual es el incentivo para que los electrones se vayan de casa. Decimos que la corriente resulta de una propiedad emergente del sistema, a la que llamamos inducción. La pregunta que nos hacemos es "¿cual es la resistencia del cable?" Por lo tanto el corolario metafórico de Moglen a la ley de Faraday dice que si enrollas la internet al rededor de cada persona en el planeta y giras al planeta, fluye el software en la red. Es una propiedad emergente de las mentes humanas conectadas el que creen cosas para el placer de unas y otras y para conquistar su incómoda sensación de estar demasiado solas. La única pregunta que hacer es: ¿cual es la resistencia de la red? El corolario metafórico de Moglen a la ley de Ohm afirma que la resistencia de la red es directamente proporcional a la fuerza del campo del sistema de la "propiedad intelectual". Así que la respuesta correcta que darle al econoenano es, resiste a la resistencia.

Por supuesto, todo esto está muy bien en teoría. "Resiste a la resistencia" suena bien, pero tendríamos un serio problema, no obstante la teoría, si los enanos estuvieran en lo correcto y nos encontráramos produciendo menos buen software porque no le permitimos a las personas poseerlo. Pero los enanos y los androides son formalistas de distintos tipos, y la ventaja del realismo es que si empiezas desde los hechos, los hechos siempre están de tu lado. Resulta que de tratar al software como propiedad privada sale mal software.

10. Ver The Virtual Scholar and Network Liberation.

II. Software como propiedad: el problema práctico

Para comprender por qué el convertir al software en propiedad produce software malo, necesitamos una introducción a la historia de este arte. De hecho, mejor comenzamos con la palabra “arte” misma. La programación de computadoras combina razonamiento específico con invención literaria.

A primera vista, estamos seguros, el código fuente parece ser una forma de composición no-literaria 11. La consideración primaria en un programa de computadora es que sea funcional, es decir, que opera de acuerdo a especificaciones que describen formalmente sus resultados en función de sus entradas. En este nivel de generalidad, lo único visible es el contenido funcional de los programas.

Pero los programas de computadoras existen como partes de sistemas de cómputo, que son colecciones interactuantes de hardware, software, y seres humanos. Los componentes humanos de un sistema de cómputo incluyen no sólo a los usuarios, sino también a las (potencialmente diferentes) personas que dan mantenimiento y que mejoran al sistema. El código fuente no sólo se comunica con la computadora que ejecuta el programa, a través del compilador intermediario que produce el código objeto en lenguaje de máquina, sino también con otros programadores.

11. Es esencial algún vocabulario básico. Las computadoras digitales en realidad ejecutan instrucciones numéricas: cadenas de bits que contienen información en el lenguaje “nativo” creado por los diseñadores de la máquina. A esto generalmente se le dice lenguaje de máquina. Los lenguajes de máquina del hardware están diseñados para una mayor velocidad en la ejecución en el nivel del hardware, y no son convenientes para su uso directo por seres humanos. Así que entre los componentes centrales de un sistema de cómputo están los lenguajes de programación, que traducen expresiones convenientes para seres humanos a lenguaje de máquina. La más común y relevante, pero de ningún modo la única forma de lenguaje de computadora es el compilador. El compilador hace traducciones estáticas, de modo que un archivo que contiene instrucciones legibles para un humano, conocidas como código fuente resulten en la generación de uno o más archivos de lenguaje ejecutable por la máquina, conocido como código de objeto.

La función del código fuente en relación con otros seres humanos no es ampliamente comprendida por los no-programadores, que tienden a pensar en los programas de cómputo como algo incomprensible. Estarían sorprendidos de saber que una gran parte de la información contenida en la mayoría de los programas es, desde el punto de vista del compilador y otros procesadores de lenguaje, comentario, esto es, material no-funcional. Los comentarios, por supuesto, están dirigidos a otras personas que posiblemente necesiten corregir un problema, o alterar, o mejorar la operación del programa. En la mayoría de los lenguajes de programación, mucho más espacio se emplea en decirle a la gente qué es lo que hace el programa que en decirle a la computadora cómo hacerlo.

El diseño de los lenguajes de programación siempre ha procedido bajo el requerimiento dual de una especificación completa a ser ejecutada por la máquina, e información descriptiva para lectores humanos. Podemos identificar tres estrategias básicas en el

diseño de lenguajes para cumplir con este propósito dual. La primera, desarrollada inicialmente con respecto al diseño de lenguajes específicos para productos particulares de hardware, colectivamente llamados ensambladores, esencialmente separaban las comunicaciones del programa en porciones para la máquina y para los humanos. Las instrucciones en ensamblador son parientes muy cercanos de las instrucciones en lenguaje-máquina: en general, una línea de un programa de ensamblador corresponde a una instrucción en el lenguaje nativo de la máquina. El programador controla la operación de la máquina en el nivel más específico posible, y (si bien disciplinado), agrega comentarios pertinentes junto con las instrucciones de la máquina, haciendo pausas cada pocos cientos de instrucciones para crear un bloque de comentarios, que proveen con un resumen de la estrategia del programa, o para documentar estructuras mayores de datos que el programa manipula.

Un segundo enfoque, ilustrado característicamente por el lenguaje COBOL (que significa Common Business Oriented Language, --Lenguaje Común Orientado a Negocios), era hacer que el programa en sí se pareciera a una serie de instrucciones en lengua natural, escritas con un estilo oscuro pero en teoría legible para humanos. Una línea de código de COBOL podría decir, por ejemplo, "MULTIPLY PRICE TIMES QUANTITY GIVING EXPANSION" (multiplica precio veces cantidad dando expansión). En un principio, cuando los expertos del Pentágono y de la industria comenzaron el diseño conjunto de COBOL en los primeros años de la década de 1960, este enfoque parecía el más prometedor. Los programas de COBOL aparentemente se documentaban a sí mismos, permitiendo tanto el desarrollo de equipos de trabajo capaces de colaborar en la creación de programas grandes, como el entrenamiento de programadores que, aunque eran trabajadores especializados, no necesitaban comprender la máquina tan íntimamente como los programas de ensamblador. Pero se eligió erróneamente el nivel de generalidad con el cual estos programas se documentaban a sí mismos. Una expresión más comprimida y en formato de fórmula de los detalles operativos, como por ejemplo, "expansión = precio x cantidad", resultaba más conveniente incluso para las aplicaciones financieras y de negocios, en donde tanto los lectores como los escritores de programas estaban acostumbrados a expresiones matemáticas, mientras que los procesos de describir tanto las estructuras de datos así como un más amplio contexto operacional del programa, no se volvían innecesarios, no obstante la verbosidad del lenguaje en que los detalles de la ejecución estaban especificados.

Por consiguiente, para el final de esa misma década los diseñadores de lenguajes comenzaron a experimentar con formas de expresión en que la mezcla de los detalles operacionales y de la información no-funcional necesaria para la modificación o reparación, era más sutil. Algunos diseñadores eligieron el camino de los lenguajes altamente simbólicos y comprimidos, en los cuales el programador manipulaba los datos de manera abstracta, de modo que "A x B" podría significar la multiplicación de dos enteros, dos números complejos, dos enormes arreglos, o cualquier otro tipo de dato capaz de algún proceso llamado "multiplicación", para ser llevada a término por la computadora con base en el contexto de las variables "A" y "B" al momento de la ejecución [12]. Puesto que este enfoque resultó en programas extremadamente concisos, se pensó, que el problema de hacer código comprensible para aquellos que luego buscarían modificarlo o repararlo se había simplificado. Al esconder los detalles técnicos de la operación de la computadora y enfatizar los algoritmos, se podían inventar lenguajes que fueran mejor que el inglés u otras lenguas naturales para la

expresión de procesos paso a paso. Los comentarios serían no sólo innecesarios sino distractores, tal como las metáforas usadas para comunicar conceptos matemáticos en inglés logran más confundir que esclarecer.

12. Este, debo decirlo, fue el camino que siguió la mayor parte de mi investigación y desarrollo, en gran parte relacionado con un lenguaje llamado APL ("A Programming Language" --Un Lenguaje de Programación) y sus sucesores. No fue, sin embargo, el enfoque finalmente dominante, por razones que serán expuestas a continuación.

Cómo creamos el enredo del microcerebro

Así que la historia de los lenguajes de programación refleja directamente la necesidad de encontrar formas de comunicación humano-máquina que fueran eficientes también para transmitir ideas complejas a lectores humanos. La "expresividad" se volvió una propiedad de los lenguajes de programación, no porque facilitara los cómputos, sino porque facilitaba la creación colaborativa y el mantenimiento de los cada vez más complejos sistemas de software.

A primera vista, esto parece justificar la aplicación del pensamiento tradicional de copyright para los trabajos resultantes. Aunque involucran sustancialmente elementos "funcionales", los programas de computadora contenían características "expresivas" de importancia capital. La doctrina del Copyright reconoció la fusión de función y expresión como algo característico de muchos tipos de trabajo registrados. El "código fuente", que contenía tanto las instrucciones para la máquina necesarias para su operación funcional como los "comentarios" expresivos dirigidos a los lectores humanos, fue un candidato apropiado para el tratamiento del Copyright.

Cierto, mientras se entienda que el componente expresivo del software estaba presente solo para facilitar la elaboración de "trabajos derivados". Si no fuera por la intención de facilitar la alteración, los elementos expresivos de los programas serían completamente innecesarios, y el código fuente no sería más copyrighteable que el código objeto, el output del procesador del lenguaje, purgado de todo excepto de las características funcionales del programa.

el estado de la industria del cómputo a lo largo de los años 60 y 70, cuando se establecieron las normas fundamentales de la programación sofisticada de computadoras, ocultaba la tensión implícita en esta situación. En ese periodo, el hardware era caro. Las computadoras eran colecciones cada vez más grandes y complejas de máquinas, y el negocio del diseño y la construcción de tal arreglo de máquinas para uso general era dominado, para no decir monopolizado, por una compañía. IBM regaló su software. Siendo más preciso, IBM era dueña de los programas que sus empleados escribían, y registraba bajo Copyright el código fuente de los mismos. Pero también distribuía los programas --incluyendo el código fuente-- a sus clientes sin cargos adicionales, y los animaba a hacer y compartir las mejoras o adaptaciones de los programas que distribuía. Para un fabricante de hardware dominante, esta estrategia tenía sentido: con mejores programas se vendían más computadoras, que era donde se basaba la rentabilidad del negocio.

Las computadoras, en este periodo, tenían tendencia a aglomerarse dentro de

organizaciones particulares, pero no a comunicarse ampliamente unas con otras. El software necesario para operar se distribuía no por medio de una red, sino con bobinas de cinta magnética. Este sistema de distribución llevaba a centralizar el desarrollo de software, por lo que aun cuando los clientes de IBM eran libres de hacer modificaciones y mejoras a los programas, esas modificaciones eran compartidas en primera instancia con IBM, la cual consideraba entonces si y cómo incorporar dichos cambios en la versión del software desarrollada y distribuida centralmente. Por lo tanto en dos modos importantes el mejor software de computadoras del mundo era libre: no costaba nada adquirirlo y los términos con los cuales se proveía, lo que permitía y alentaba la experimentación, los cambios, y las mejoras [13]. Que el software en cuestión fuera propiedad de IBM bajo la ley de Copyright prevaleciente ciertamente establecía ciertos límites teóricos respecto de la habilidad de los usuarios para distribuir a terceros sus mejoras o adaptaciones, pero en práctica el software de mainframes era desarrollado cooperativamente por el fabricante de hardware dominante y sus usuarios técnicamente sofisticados, empleando los recursos de distribución del fabricante para propagar las mejoras resultantes gracias a la comunidad de los usuarios. El derecho de excluir a terceros, una de las más importantes "varas en el atado" de los derechos de propiedad (en una metáfora amada por la Corte Suprema de los Estados Unidos), prácticamente no tenía importancia, o era incluso indeseable, en el corazón mismo del negocio del software [14].

13. Esta descripción deja de lado algunos detalles. Para mediados de los años 70 IBM había adquirido competencia significativa en el negocio de las supercomputadoras, al mismo tiempo que el gran juicio antimonopolista llevado en contra de dicha compañía por el gobierno de los E.U. condujo a la decisión de "separar", o cobrar por separado el software. En este sentido de menor importancia el software dejó de ser libre. Pero --sin entrar en la ahora muerta pero entonces acalorada controversia respecto de las políticas de precios del software de IBM--, la revolución de la "separación" tuvo menores efectos en las prácticas sociales de la manufactura de software de los que se hubieran podido suponer. Como uno de los colegas responsables de las mejoras técnicas de un lenguaje de programación de IBM entre 1979 y 1984, por ejemplo, fui capaz de tratar tal producto como "casi libre", es decir, de discutir con los usuarios los cambios que ellos propusieran o hicieran a los programas, y de involucrarme con ellos en el desarrollo cooperativo del producto para beneficio de todos los usuarios.

14. Esta descripción está sumamente comprimida, y le parecerá tanto demasiado simplificada como indebidamente rosada a quienes también trabajaron en la industria durante ese periodo de su desarrollo. La protección con copyright del software de computadora fue un tema controvertido en los años 70, lo que llevó a la famosa comisión CONTU (Commission on New Technological Uses of Copyrighted Works -- Comisión para los Nuevos modos de Uso Tecnológico de Trabajos con Copyright) y sus blandas recomendaciones pro-copyright de 1979. IBM parecía mucho menos cooperativa para con sus usuarios en ese entonces de lo que este esbozo describe. Pero el elemento más importante es el contraste con el mundo creado por las PCs, la internet, y el dominio de Microsoft, con el ímpetu resultante del movimiento del software libre, y estoy aquí concentrándome en las características que expresan dicho contraste.

Después de 1980, todo fue diferente. El mundo del hardware de las supercomputadoras

cedió el paso en un lapso de diez años al mundo de las PCs como bienes de consumo. Y, como una contingencia en el desarrollo de la industria, el único y más importante elemento del software que se ejecutaba en ese bien de consumo, el sistema operativo, se volvió el único producto significativo de una compañía que no hacía hardware. El software básico de alta calidad dejó de ser parte de la estrategia de diferenciación de producto de los fabricantes de hardware. En cambio, una compañía con una porción avasalladora del mercado, y con la común ausencia semi-monopolista de interés en fomentar la diversidad, fijó las prácticas de la industria del software. En tal contexto, el derecho para excluir a otros de la participación en la formación del producto, se volvió profundamente importante. El poder de Microsoft en el mercado se fundaba completamente en su propiedad privada del código fuente de Windows.

Para Microsoft, que otros hicieran "trabajos derivados", también conocidos como reparaciones y mejoras, amenazaba el activo central de su negocio. De hecho, la tendencia que establecen procesos judiciales subsecuentes, es que la estrategia de negocios de Microsoft fue encontrar ideas innovadoras en otras partes del mercado del software, comprarlas y ya sea suprimirlas o incorporarlas en su producto privativo. Mantener el control de la operación básica de las computadoras fabricadas, vendidas, compradas y usadas por otros representó una influencia profunda y redituable sobre el desarrollo de la cultura [15]; el derecho a excluir regresó al escenario principal en el concepto del software como propiedad.

15. Discuto la importancia del software de PC en este contexto, la evolución del "mercado para globos oculares" y "la vida patrocinada" en otros capítulos de mi próximo libro, La barbacoa invisible, del cual este ensayo es parte.

El resultado, en lo que concierne a la calidad del software, fue desastroso. El monopolio era una corporación rica y poderosa que empleaba un gran número de programadores, pero no podía de ningún modo permitirse el número de evaluadores, diseñadores, y desarrolladores necesarios para producir el software flexible, robusto y técnicamente innovador apropiado para el vasto arreglo de condiciones en que operaban computadoras personales cada vez más ubicuas. Su estrategia de marketing fundamental comportaba diseñar su producto para los usuarios menos sofisticados técnicamente, y usar "miedo, incertidumbre, y duda" (FUD, por sus siglas en inglés) para alejar a los usuarios sofisticados de competidores potenciales, cuya supervivencia a largo plazo frente al poder de Microsoft en el mercado siempre estaba en duda.

Sin la interacción constante entre los usuarios capaces de reparar y mejorar, y el fabricante del sistema operativo, el inevitable deterioro de calidad no pudo detenerse. Pero puesto que la revolución de las PCs expandió el número de usuarios exponencialmente, prácticamente cualquiera de los que entraron en contacto con los sistemas resultantes no tuvo nada con qué compararlos. Inconscientes de los estándares de estabilidad, confiabilidad, mantenibilidad y efectividad que habían sido establecidos previamente en el mundo de las supercomputadoras, no se podía esperar de los usuarios de las computadoras personales que entendieran qué tan mal, en términos relativos, el monopolio del software funcionaba. Al tiempo que el poder y la capacidad de las computadoras personales se expandía rápidamente, los defectos del software se volvieron menos obvios en medio del incremento general de productividad. Los usuarios ordinarios, más de la mitad de los cuales estaban asustados de una

tecnología que no entendían prácticamente para nada, acogieron de hecho con buenos ojos los defectos del software. En una economía que experimentaba transformaciones misteriosas, con la desestabilización concurrente de millones de carreras, era tranquilizador, en un modo perverso, que ninguna computadora personal parecía capaz de funcionar por más de unas cuantas horas consecutivas sin descomponerse. Aun cuando era frustrante perder parte del trabajo empezado cada vez que una falla innecesaria ocurría, la falibilidad evidente de las computadoras era intrínsecamente reconfortante [16].

16. Este mismo patrón de ambivalencia, en el cual la mala programación conducente a la inestabilidad generalizada de la nueva tecnología es simultáneamente temible y reconfortante para los técnicamente incompetentes, puede verse también en el fenómeno principalmente norteamericano de la histeria del año 2000.

Nada de esto era necesario. La baja calidad del software para computadoras personales pudo haberse revertido incluyendo a los usuarios en el proceso inherentemente revolucionario del diseño y la implementación del software. Un modelo Lamarckiano, en el cual las mejoras pudieran hacerse en cualquier lugar, por parte de cualquiera, y heredadas por todos los demás, hubiera liquidado el déficit, restaurando para el mundo de las PCs la estabilidad y confiabilidad del software que se hacía en el ambiente cuasi-privativo de la era de las supercomputadoras. Pero el modelo de negocios de Microsoft impedía la herencia Lamarckiana de mejoras en el software. La doctrina del Copyright, en general y tal como se aplica al software en particular, predispone al mundo hacia el creacionismo; y en esta instancia, el problema es que BillG, el creador, estaba lejos de la infalibilidad, y de hecho ni siquiera estaba intentando conseguirla.

Para agudizar la ironía, el crecimiento de la red convirtió a la alternativa no-privativa incluso más práctica. Lo que los escritos tanto académicos como populares denominan como una cosa ("la internet") es de hecho el nombre de una condición social: el hecho de que todos en la sociedad de la red estén conectados directamente, sin intermediarios, con todos los demás [17]. La interconexión global de redes eliminó el cuello de botella que requería que un fabricante centralizado de software racionalizara y distribuyera los resultados de las innovaciones individuales en la era de las supercomputadoras.

Y así, en una de las pequeñas ironías de la historia, el triunfo global del mal software en la era de las PCs fue revertido por una sorprendente combinación de fuerzas: la transformación social iniciada por la red, una teoría europea hacía tiempo desechada de economía política, y una pequeña banda de programadores al rededor del mundo movilizada por una simple idea.

17. Las implicaciones críticas de esta simple observación acerca de nuestras metáforas está desarrollada en "How Not To Think about 'The Internet'", en The Invisible Barbecue, próximo a publicarse.

El software quiere ser libre; o Cómo dejamos de preocuparnos y aprendimos a amar la Bomba

Mucho antes de que la red de redes fuera una realidad práctica, incluso antes de que fuera una aspiración, había el deseo de que las computadoras operaran con base en

software disponible libremente para cualquiera. Esto empezó como una reacción en contra de el software privativo en la era de las supercomputadoras, y requiere otra breve divagación.

Aun cuando IBM era el mayor vendedor de computadoras de propósito general en la era de las supercomputadoras, no era le mayor diseñador ni fabricante de dicho hardware. El monopolio telefónico, American Telephone & Telegraph (AT&T), era de hecho más grande que IBM, pero consumía sus productos internamente. Y en los famosos Laboratorios Bell que eran el brazo de investigación del monopolio telefónico, en los años 60 tardíos, los desarrollos en lenguajes computación dieron a luz a un sistema operativo llamado Unix.

La idea detrás de Unix fue crear un sólo sistema operativo escalable para operar en todas las computadoras, de pequeñas a grandes, que el monopolio telefónico fabricaba para sí mismo. Lograr ese objetivo significaba escribir un sistema operativo ya no en lenguaje de máquina, ni en ensamblador cuya forma lingüística estaba relacionada con un diseño particular de hardware, sino en un lenguaje más expresivo y generalizado. El lenguaje elegido fue también un invento de los laboratorios Bell llamado "C" [18]. El lenguaje C se volvió común, incluso dominante, para muchos tipos de tareas de programación, y para los final de los años 70 el sistema operativo Unix escrito en ese lenguaje había sido transferido a computadoras hechas por muchos fabricantes y con muchos diseños.

18. Los lectores técnicos observarán nuevamente que esto comprime eventos que se desarrollaron entre 1969 y 1973.

AT&T distribuyó Unix ampliamente, y debido al diseño mismo del sistema operativo, tuvo que hacer dicha distribución en código fuente de C. Pero AT&T retuvo la propiedad del código fuente e instaba a los usuarios a comprar licencias que prohibían tanto su redistribución como el hacer trabajos derivados. Grandes centros de cómputo, ya sea industriales o académicos, podían permitirse comprar dichas licencias, pero los individuos no, y las restricciones de la licencia evitaban que la comunidad de programadores que usaban Unix lo mejoraran de un modo evolutivo, en vez de un modo episódico. Y mientras los programadores al rededor del mundo empezaron a aspirar e incluso esperar una revolución de las computadoras personales, el estatus "no libre" de Unix se volvió una fuente de preocupación.

Entre 1981 y 1984, un hombre ideó una cruzada para cambiar la situación. Richard M. Stallman, entonces un empleado del laboratorio de inteligencia artificial del MIT concibió el proyecto del rediseño e implementación independiente y colaborativo de un sistema operativo que habría de ser software libre de verdad. En términos del mismo Stallman, el software libre debía ser un asunto de libertad, no de precio. Cualquiera debía poder modificar y redistribuir libremente dicho software, o venderlo, con la sola restricción de que quien lo hiciera no intentara reducir esos mismos derechos a quienes lo recibieran. De este modo el software libre podía volverse un proyecto auto-organizado, en el cual ninguna innovación se perdiera por culpa del ejercicio de los derechos de propiedad. Ese sistema, por decisión de Stallman, sería llamado GNU, lo que significaba (en un ejemplo inicial del gusto por los acrónimos recursivos que ha caracterizado desde entonces al software libre), "GNU No es Unix". A pesar del recelo

respecto del diseño fundamental de Unix, así como respecto de sus condiciones de distribución, GNU pretendía beneficiarse de la amplia aunque no-libre distribución de Unix. Stallman empezó el proyecto GNU escribiendo componentes del eventual sistema que también estaban diseñados para funcionar sin modificaciones en los sistemas Unix pre-existentes. El desarrollo de las herramientas GNU podía de ese modo hacerse directamente en el ambiente de los centros de cómputo avanzados de universidades y otras instituciones alrededor del mundo.

La escala de dicho proyecto era inmensa. De algún modo se tenían que encontrar, organizar, y poner a trabajar a programadores voluntarios, en el desarrollo de todas las herramientas necesarias para la construcción final. Stallman mismo fue el autor principal de varias herramientas fundamentales. Otras fueron contribución de pequeños o grandes equipos de programadores en otros lados, y asignados al proyecto de Stallman o distribuidos directamente. Algunas locaciones en distintos puntos de la red que estaba desarrollándose se volvieron archivos para el código fuente de estos componentes GNU, y a lo largo de los años 80 las herramientas GNU lograron reconocimiento y aceptación por parte de los usuarios de Unix al rededor del mundo. La estabilidad, confiabilidad, y mantenibilidad de las herramientas GNU se volvieron emblemáticas, mientras las profundas habilidades de Stallman como diseñador siguieron adelantándose a, y proveyendo metas para, el proceso que estaba evolucionando. El premio otorgado a Stallman por la fraternidad MacArthur en 1990 fue un reconocimiento apropiado de sus innovaciones conceptuales y técnicas así como de sus consecuencias sociales.

El proyecto GNU, y la Fundación para el software libre a la que dio vida en 1985, no fueron las únicas fuentes de ideas respecto al software libre.Varias formas de licencias de copyright diseñadas para impulsar el software libre, o parcialmente libre, empezaron a desarrollarse en la comunidad académica, en su mayor parte relacionadas con ambiente Unix. La universidad de California en Berkeley empezó el diseño e implementación de otra versión de Unix para su distribución libre dentro de la comunidad académica. Unix BSD, como llegó a ser conocido, también tomó como estándar de diseño el Unix de AT&T. El código fuente fue divulgado ampliamente y constituyó un depósito de herramientas y técnicas, pero los términos de su licencia limitaban el rango de su aplicabilidad, mientras que la eliminación del código privativo para elementos de hardware específicos significaba que nadie podía compilar de hecho un sistema operativo funcional para ninguna computadora en particular a partir de BSD. Otros trabajos universitarios resultaron también en software quasi-libre; la interfaz de usuario gráfica (o GUI) para sistemas Unix llamada X Windows, por ejemplo, fue creada en el MIT y distribuida junto con el código fuente, permitiendo modificaciones libres. Y en 1989-1990, un estudiante de ciencias de la computación en la universidad de Helsinki, Linus Torvalds, empezó el proyecto que cerró el círculo y energizó por completo la visión del software libre.

Lo que Torvalds hizo fue empezar a adaptar una herramienta de enseñanza de ciencias de la computación para su uso en el mundo real. El kernel MINIX de Andrew Tannenbaum [19], era un legado de varios cursos de sistemas operativos, con ejemplos de soluciones básicas a problemas básicos. Lentamente, y al principio sin reconocer su intención, Linus empezó a convertir el kernel MINIX en un kernel de facto para UNIX en procesadores Intel x86, que son los motores dentro de las computadoras personales de

consumo del mundo. Mientras Linus iniciaba el desarrollo de dicho kernel, al que nombró Linux, se dio cuenta de que el mejor modo para que su proyecto funcionara era ajustar sus decisiones de diseño de manera que los componentes GNU existentes fueran compatibles con su kernel.

El resultado del trabajo de Torvalds fue el lanzamiento a la red en 1991 de un modelo incompleto pero funcional de un kernel en software libre para un sistema operativo tipo Unix en PCs, totalmente compatible y diseñado de modo convergente con el conjunto enorme y de alta calidad de componentes creados por el proyecto GNU de Stallman y distribuidos por la Fundación del software libre. Debido a que Torvalds eligió lanzar el kernel Linux registrado bajo la Licencia Pública General de la Fundación para el Software Libre, de la cual hablaremos más abajo, los cientos y eventualmente miles de programadores alrededor del mundo que decidieron contribuir con sus esfuerzos para el desarrollo ulterior del kernel podían estar seguros de que sus aportaciones resultarían en software libre permanente que nadie podría convertir en un producto privativo. Cualquiera sabía que cualquier otro individuo podía probar dichas aportaciones, mejorarlas, y redistribuir sus mejoras. Torvalds aceptó las contribuciones gratuitamente, y con un estilo genialmente efectivo mantuvo el rumbo global sin apagar el entusiasmo. El desarrollo del kernel Linux probó que la Internet hacía posible sumar equipos de programadores mucho más grandes de lo que ningún fabricante comercial podía pagar, unidos casi sin jerarquías en un proyecto de desarrollo que involucraba en última instancia más de un millón de líneas de código de computadora -una escala de colaboración entre voluntarios no pagados geográficamente dispersos previamente inimaginable en la historia de la humanidad [20].

19. Los sistemas operativos, incluido Windows (que esconde el hecho de sus usuarios tanto como le es posible), son de hecho colecciones de componentes, y no unidades enteras. La mayor parte de lo que un sistema operativo hace (administrar archivos, controlar la ejecución de procesos, etc.) puede abstraerse a partir de los detalles del hardware en el que el sistema operativo se ejecuta. Sólo un pequeño núcleo interior del sistema debe tratar con las peculiaridades excéntricas de cierto hardware particular. Una vez que el sistema operativo se ha escrito en un lenguaje general como C, sólo ese núcleo interior, conocido en el negocio como el kernel, será altamente específico respecto a la arquitectura de ciertas computadoras en particular.

20. Un análisis cuidadoso y creativo de cómo Torvalds hizo funcionar este proceso, y lo que implica para las prácticas sociales de desarrollo de software, fue provisto por Eric S. Raymond en su ensayo fundamental de 1997, La Catedral y el Bazaar, el cual jugó un rol significativo en la expansión de la idea del software libre.

Para 1994, Linux había llegado a su versión 1.0, lo que representaba un kernel de producción usable. El nivel 2.0 se alcanzó en 1996, y para 1998, con el kernel en la versión 2.2.0 y ya disponible no sólo para máquinas x86 sino para una variedad de otras arquitecturas, GNU/Linux --la combinación del kernel Linux y el conjunto mucho mayor de componentes del proyecto GNU-- y Windows NT eran los únicos dos sistemas operativos en el mundo que estaban incrementando su cuota de mercado. Una evaluación interna de Microsoft sobre la situación, filtrada en octubre de 1998 y posteriormente reconocida por dicha compañía como genuina, concluía que "Linux representa lo mejor en su clase de Unix, el cual es confiable en aplicaciones críticas, y

--debido a su código abierto (sic)-- tiene una credibilidad de largo plazo que supera a muchos otros sistemas operativos competitivos." [21] Los sistemas GNU/Linux ahora son usados alrededor del mundo para operar de todo: desde servidores web en importantes sitios de comercio electrónico hasta clústers de supercomputadoras ad-hoc o la infraestructura de redes de bancos. GNU/Linux se encuentra en el transbordador espacial y corriendo tras bambalinas en computadoras de (sí) Microsoft. Las evaluaciones de la industria respecto de la confiabilidad comparativa de los sistemas Unix han mostrado repetidamente que Linux es por mucho el kernel de Unix más confiable y estable, con una confiabilidad superada solamente por las herramientas GNU mismas. GNU/Linux no sólo supera a las versiones comerciales privativas de Unix para PCs en las mediciones, sino que es renombrado por su habilidad para funcionar, sin molestias y sin quejas, por meses y meses en ambientes de grandes volúmenes de información y alto rendimiento sin romperse.

Otros componentes del movimiento del software libre han sido igualmente exitosos. Apache, por mucho el servidor web líder en el mundo, es software libre, igual que Perl, el lenguaje de programación que es la lingua franca de los programadores que construyen sitios web sofisticados. Netscape Communications ahora distribuye su navegador Netscape Communicator 5.0 como software libre, registrado con una variante cercana de la Licencia Pública General de la Fundación para el Software Libre. Grandes fabricantes de PCs, incluyendo a IBM, ya anunciaron sus planes (o ya están distribuyendo) a GNU/Linux como una opción para sus clientes en sus mejores PCs, para su uso como servidores web o servidores de archivos. Samba, un programa que le permite a las computadoras con GNU/Linux actuar como servidores de archivos Windows NT, es usado alrededor del mundo como una alternativa a los servidores Windows NT, y provee una competencia efectiva de bajo costo para Microsoft en su propio mercado doméstico. Debido a los estándares de calidad del software que han sido reconocidos en la industria por décadas --y cuya continua relevancia estará clara para ti la próxima vez que tu PC con windows truene-- las noticias al final del siglo no son ambiguas. La corporación más rentable y poderosa del mundo va en un distante segundo lugar, tras haber sacado a todos excepto al vencedor verdadero de la carrera. El propietarismo unido al vigor capitalista destruyó a la competencia comercial significativa, pero cuando se trataba de hacer buen software, el anarquismo venció.

21. Esta es una cita de lo que se conoce en el negocio como el "Halloween memo", el cual puede encontrarse, como fue comentado por Eric Raymond, a quien se lo filtraron, en http://www.opensource.org/halloween/halloween1.html

III. El anarquismo como un modo de producción

Es una linda historia, y si tan sólo el iPdroide y el econoenano no hubieran sido cegados por la teoría, la hubieran visto llegar. Pero aun cuando algunos de nosotros hemos estado trabajando en ello y prediciéndolo por años, las consecuencias teóricas son tan subversivas para las formas de pensar que mantienen a nuestros enanos y androides cómodos que difícilmente se les puede culpar por rechazar verla. Los hechos probaron que algo estaba mal con la metáfora de los "incentivos" que es la base de los razonamientos convencionales sobre la propiedad intelectual [22]. Pero los hechos hicieron más. Nos dieron un vistazo inicial al futuro de la creatividad humana en un mundo de interconexión global, y no es un mundo hecho para enanos o androides.

Mi argumentación, antes de hacer una pausa para refrigerios en el mundo real, puede resumirse de este modo: El software --ya sean programas ejecutables, música, arte visual, liturgias, armamento, o lo que sea-- consiste de cadenas de bits, las cuales aun cuando son esencialmente indistinguibles entre sí son tratadas por una multiplicidad confusa de categorías legales. Dicha multiplicidad es inestable a largo plazo por razones integrales al proceso legal. La diversidad inestable de reglas es ocasionada por la necesidad de distinguir entre los tipos de intereses de propiedad hechos con cadenas de bits. Esta necesidad es sentida principalmente por aquellos que se sostienen de lucrar de las formas aceptables socialmente de monopolio creadas por el hecho de tratar a las ideas como propiedad privada. Aquellos de nosotros que estamos preocupados por la inequidad social y hegemonía cultural creadas por este régimen insatisfactorio intelectualmente y moralmente repugnante somos abucheados. Quienes nos abuchean, los enanos y los androides, creen que esas reglas de propiedad son necesarias no debido a un anhelo descarado de vivir en Murdochworld --aunque un poco de cooptación lujosa es siempre bienvenida-- sino porque la metáfora de los incentivos, a la cual toman no sólo como un simple símbolo sino como un fundamento, prueba que esas reglas --a pesar de sus lamentables consecuencias- son necesarias si queremos hacer buen software. La única forma para seguir creyendo eso es ignorando los hechos. Al centro de la revolución digital, con las cadenas de bits ejecutables que hacen todo lo demás posible, los regímenes propietaristas no sólo no mejoran las cosas, sino que pueden empeorarlas radicalmente. Los conceptos de propiedad, y cualquier otra cosa que pueda estar mal con ellos, no facilitan y de hecho han retrasado el progreso.

¿Pero cual es esta misteriosa alternativa? El software libre existe, pero ¿cuales son sus mecanismos, y cómo se generaliza hacia una teoría no propietarista de la sociedad digital?

22. Hace tan poco como 1994 un estudioso de leyes y economía talentoso y técnicamente competente (aunque usuario de windows) en una de las principales escuelas de leyes de los E.U. me informó confiadamente que el software libre no podía existir, porque nadie tendría ningún incentivo para hacer programas realmente sofisticados que requirieran una inversión sustancial de esfuerzo tan sólo para regalarlos.

La teoría legal del software libre

Existe un mito, como la mayoría de los mitos parcialmente fundado en la realidad, de

que los programadores de computadoras son todos libertarios. Los de derecha son capitalistas, devotos de sus acciones de bolsa, desdeñan los impuestos, los sindicatos y las leyes de derechos civiles; los de izquierda odian al mercado y a todos los gobiernos, creen en el cifrado extremo sin importar cuanto terrorismo nuclear pueda ocasionar[23], y les disgusta Bill Gates porque es rico. Hay sin duda fundamentos para esas creencias. Pero la diferencia más significativa entre el pensamiento político dentro del mundillo de los digitalmente alfabetizados y fuera de él es que en la sociedad en red, el anarquismo (o más propiamente, el individualismo anti-posesivo) es una filosofía política viable.

El núcleo del éxito del movimiento del software libre, y el mayor logro de Richard Stallman, no es un fragmento de código de computadora. El éxito del software libre, que incluye el abrumador éxito de GNU/Linux, es resultado de la habilidad para aprovechar cantidades extraordinarias de esfuerzos de alta calidad para proyectos de tamaños inmensos y profunda complejidad. Y esta habilidad a su vez resulta del contexto legal en el cual la fuerza laboral es movilizada. Como diseñador visionario Richard Stallman creó más que Emacs, GDB, o GNU. Creó la Licencia pública general.

23. Este asunto también merece un escrutinio especial, incrustado como está con una defensa especial del lado del poder estatal. Lean mi breve ensayo "So Much for Savages: Navajo 1, Government 0 in Final Moments of Play."

La GPL [24], también conocida como el copyleft, usa el copyright, parafraseando a Toby Milsom, para simular el fenómeno del anarquismo. Como el preámbulo de la licencia lo dice:

Cuando hablamos de software libre, nos estamos refiriendo a libertad, no al precio. Nuestras Licencias Públicas Generales están diseñadas para garantizar que tengas la libertad de distribuir copias del software libre (y cobrar por ese servicio si así lo deseas), que recibas el código fuente o puedas obtenerlo si lo deseas, que puedas cambiar el software o usar pedazos de él en nuevos programas libres; y que sepas que puedes hacer todas esas cosas.

Para proteger tus derechos, necesitamos poner restricciones que prohíben a cualquiera negarte esos derechos o pedirte que cedas esos derechos. Estas restricciones se traducen en ciertas responsabilidades para ti si distribuyes copias del software, o si lo modificas.

Por ejemplo, si distribuyes copias de un programa libre, ya sea gratis o por una tarifa, debes darle a los que lo reciban todos los derechos que tú tienes. Debes asegurarte de que ellos, también, reciban o puedan obtener el código fuente. Y debes mostrarles estos términos para que conozcan sus derechos.

Muchas variantes de esta idea básica del software libre han sido expresadas en licencias de varios tipos, como ya he indicado. La GPL es diferente de las demás formas de expresar dichos valores en un aspecto crucial. En la sección 2 de la licencia se proporcionan los detalles en una parte pertinente:

Puedes modificar tu copia o copias del programa o cualquier porción del mismo,

formando así un trabajo basado en el programa, y copiar y distribuir dichas modificaciones o trabajo..., siempre que también cumplas con las siguientes condiciones:

...

24. Lee la Licencia Pública General, Versión 2, junio 1991.

b) Debes hacer que cualquier trabajo que distribuyas o publiques, que en total o en parte contenga o sea derivado del Programa o de cualquier parte del mismo, sea registrado en su totalidad, sin cargos para todos los terceros, bajo los términos de esta licencia.

La sección 2(b) de la GPL es algunas veces llamada "restrictiva", pero su intención es liberadora. Crea un bien común al que todos pueden añadir pero al cual nadie puede restarle. Debido a la S2(b), cada contribuyente a un proyecto registrado con la GPL tiene garantizado que él, y todos los demás usuarios, también serán capaces de ejecutar, modificar y redistribuir el programa indefinidamente, que el código fuente siempre estará disponible, y que, a diferencia del software comercial, su longevidad no podrá ser limitada por las contingencias del mercado o las decisiones de futuros desarrolladores. Esta "herencia" de la GPL ha sido criticada algunas veces como un ejemplo de la inclinación anti-comercial del movimiento del software libre. Nada podría estar más lejos de la verdad. El efecto de la S2(b) es hacer que los distribuidores comerciales de software libre sean mejores competidores contra los negocios de software privativo. Para confirmar este punto, no se puede hacer nada mejor que preguntarle a los competidores privativos. En los términos que el autor del memo "Halloween" de Microsoft, Vinod Vallopillil, lo puso:

La GPL y su aversión a la división del código asegura a los clientes que ellos no están metidos en un callejón sin salida evolutivo al suscribir a una versión comercial particular de Linux.

El callejón sin salida evolutivo es el meollo del argumento FUD del software [25].

25. V. Vallopillil, Software de código abierto: Una (¿nueva?) Metodología de desarrollo.

Traducido de la Micro-habla, esto quiere decir que la estrategia por medio de la cual el fabricante privativo dominante aleja a los clientes de su competencia --infundiendo miedo, incertidumbre y duda respecto de la viabilidad a largo plazo de otros softwares-- no es efectiva con respecto a los programas registrados con la GPL. Los usuarios del código GPL, incluyendo aquellos que compran el software y los sistemas a algún revendedor comercial, saben que las mejoras y reparaciones futuras estarán accesibles en los bienes comunes, y no deben temer ni que desaparezca su proveedor actual ni que alguien use una mejora particularmente atractiva o una reparación desesperadamente necesitada para impulsar la "privatización del programa".

Este uso de las reglas de propiedad intelectual para crear bienes comunes en el ciberespacio es la estructura institucional central que permite el triunfo anarquista. Asegurar el libre acceso y permitir modificaciones a cada etapa del proceso significa que

la evolución del software ocurre de un rápido modo Lamarckiano: cada característica favorable adquirida del trabajo de los otros puede ser heredada directamente. De ahí la velocidad con la que el kernel de Linux, por ejemplo, creció por encima de todos sus predecesores privativos. Puesto que la deserción es imposible, los gorrones son bienvenidos, lo cual resuelve uno de los rompecabezas centrales de la acción colectiva en un sistema social privativo.

La producción no privativa es también directamente responsable de la famosa estabilidad y confiabilidad del software libre, la cual surge de la que Eric Raymond llama "la ley de Linus": con suficientes ojos, todos los errores son superficiales. En términos prácticos, el acceso al código fuente significa que si tengo un problema puedo arreglarlo. Pero puesto que puedo arreglarlo, casi nunca tengo que hacerlo, porque alguien más casi siempre ya lo vio y arregló antes.

Para la comunidad del software libre, el compromiso con la producción anarquista debe ser un imperativo moral; porque como escribió Richard Stallman, esto es acerca de la libertad, no del precio. O podría ser un asunto de utilidad, buscar producir mejor software del que los modos de trabajo privativos puedan permitir. Desde el punto de vista del androide, el copyleft representa la perversión de la teoría, pero resuelve de mejor manera que cualquier otra propuesta a lo largo de las pasadas décadas el problema de aplicarle copyright a las inextricablemente fusionadas características funcional y expresiva de los programas de computadora. Que el copyleft produzca mejor software que la alternativa no implica que los principios del copyright tradicionales debieran estar prohibidos para aquellos que quieran ser dueños y llevar al mercado productos de software inferiores, o (más caritativamente) para aquellos cuyos productos tengan un atractivo demasiado estrecho para la producción comunal. Pero nuestra historia debería servirle de advertencia a los androides: el mundo del futuro tendrá muy poca relación con el mundo del pasado. Las reglas en el presente están siendo dobladas en dos direcciones. Los dueños corporativos de los "iconos culturales" y de otros bienes, quienes buscan cada vez mayores plazos para los autores corporativos, convirtiendo el "tiempo limitado" del artículo I, §8 en una posesión irrestricta, naturalmente han estado silbando música en los oídos de los androides[26]. Después de todo, ¿quién le compró a los androides sus boletos para el concierto? Pero mientras que la posición propietarista busca incrustarse cada vez más fuertemente en una concepción del copyright liberada de las molestias menores de los lapsos limitados y el uso justo, en el centro mismo de nuestro sistema de "software cultural", el contra-ataque anarquista ha empezado. Peores cosas les ocurrirán a los androides, como veremos. Pero primero, debemos pagar nuestra última deuda con los enanos.

26. La aproximación de la expiración de la propiedad de Mickey Mouse por parte de Disney requiere, desde el punto de vista de ese rico "contribuidor a las campañas políticas", por ejemplo, una alteración de la ley general de copyright de los Estados Unidos. Ver "Not Making it Any More? Vaporizing the Public Domain" en el libero por salir La Barbacoa invisible.

Porque ahí está: el imán de Faraday y la creatividad humana

Después de todo, merecen una respuesta. ¿Porqué hace la gente software libre si no obtiene lucros? Dos respuestas se han dado comúnmente. Una es mitad cierta y la otra

está equivocada, pero ambas son insuficientemente simples.

La respuesta equivocada está incrustada en numerosas referencias a "la cultura hacker del intercambio de regalos". Este uso de jerga etnográfica deambuló en el ramo hace algunos años y se volvió rápidamente, aunque engañosamente, ubicua. Nos recuerda sólo que los econometristas han corrompido tanto nuestros procesos de pensamiento que cualquier tipo de comportamiento económico que no sea de mercado les parece igual al de cualquier otro tipo. Pero el intercambio de regalos, como el trueque en un mercado, es una institución propietarista. La reciprocidad es central para estas representaciones simbólicas de dependencia mutua, así, si las papas o los pescados son mal pesados, surgen problemas. El software libre, arriesgando repetirme, es un bien común: ningún ritual de reciprocidad es representado aquí. Pocas personas entregan código que otros venden, usan, cambian o prestan a granel para echar a andar partes de otras cosas. No obstante el gran número de personas (decenas de miles, por mucho) que han contribuido a GNU/Linux, este es órdenes de magnitud menor que el número de usuarios que no hacen ninguna contribución de ningún tipo [27].

27. Un estimado reciente de la industria pone el número de sistemas operando con Linux mundialmente en 7.5 millones. Ver Josh McHugh, 1998. "Linux: Haciendo el hack global", Forbes (agosto 10). Porque el software se obtiene libremente por medio de internet, no hay un modo simple de valorar su uso real.

Una parte de la respuesta correcta es sugerida por la afirmación de que el software libre es hecho por aquellos que buscan compensaciones a su reputación debido a sus actividades. Los hackers famosos de Linux, dice esta teoría, son conocidos por todo el planeta como deidades de la programación. De ello se deriva ya sea una auto-estima ampliada o mejoras materiales indirectas [28]. Pero las deidades de la programación, por mucho que hayan contribuido al software libre, no han hecho el grueso del trabajo. Las reputaciones, como Linus Torvalds mismo ha señalado con frecuencia, están hechas del reconocimiento voluntario de que todo fue hecho por alguien más. Y, como muchos observadores han notado, el movimiento del software libre también ha producido excelente documentación. Escribir documentación no es lo que hacen los hackers para alcanzar a ser cool, y mucha de la documentación fue escrita por personas distintas de los que escribieron el código. Y tampoco debemos limitar las ventajas materiales indirectas de la autoría a incrementos en el capital de la reputación. La mayoría de los autores de software libre que conozco tienen trabajos diurnos en industrias de tecnología, y las habilidades en que se afinan para los trabajos más creativos que hacen fuera del mercado laboral sin duda algunas veces incrementan palpablemente su valor dentro del. Y así, conforme los productos de software libre ganaron masa crítica y se volvieron la base de todo un conjunto de modelos de negocios construidos al rededor de distribuir comercialmente aquello que la gente puede también obtener por nada, un número cada vez mayor de personas han sido empleadas específicamente para escribir software libre. Pero para poder ser empleable en ese campo de trabajo, deben haberse establecido previamente en él. Llanamente, pues, este motivo está presente, pero no lo explica todo.

De hecho, el resto de la respuesta es demasiado simple para haber recibido la consideración adecuada. La mejor forma de entender es revisar la breve y poco cantada carrera de un autor de software libre tacaño inicialmente. Vinod Vallopillil de Microsoft,

mientras escribía el análisis competitivo de Linux que se filtró como el segundo de los famosos "memorandos Halloween", compró e instaló un sistema Linux en una de sus computadoras en la oficina. Tuvo problemas porque la distribución (comercial) de Linux que instaló no tenía un mecanismo para manejar el protocolo DHCP de asignación de direcciones IPs dinámicas. El resultado fue suficientemente importante para que nos arriesguemos nuevamente a exponernos largo rato al estilo de redacción de Microsoft:

28. Eric Raymond es un partisano de la teoría del "aumento del ego", a la cual añade otra comparación etnográficamente falsa, de la composición de software libre con la potlatch Kwakiutl. Ver Eric S. Raymond, 1998. Cultivando la noosfera. Pero la potlatch, ciertamente una forma de competencia de estatus, es diferente del software libre por dos razones fundamentales: es esencialmente jerárquico, y el software libre no lo es, y, como supimos desde que Thorstein Veblen llamó la atención en un inicio respecto de su importancia, es una forma de desperdicio conspicuo. Ver Thorstein Veblen, 1967. La teoría de la clase del ocio. New York: Viking, p. 75. Esas son precisamente las bases que distinguen a la cultura anti-jerárquica y utilitaria del software libre de sus contrapartes propietarias.

Un pequeño número de sitios web y FAQs después, encontré un sitio FTP con un cliente de DHCP para Linux. El programa DHCP fue desarrollado por un ingeniero empleado por Fore Systems (como lo evidenciaba su dirección de correo electrónico; creo, sin embargo, que fue desarrollado en su tiempo libre). Un conjunto adicional de documentación/manuales fue escrito para el cliente de DHCP por un hacker en Hungría que proveía instrucciones relativamente simples respecto de cómo instalar y ejecutar el programa.

Bajé y descomprimí el cliente y tecleé dos simples comandos:

Make - compila los binarios del cliente
Make install - instaló los binarios como un mecanismo de Linux

Teclear "DHCPCD" (o DHCP Client Deamon) en la línea de comandos ejecutaba el proceso de búsqueda de DHCP y voila, ya tenía una IP de red funcional.

Puesto que recién había descargado el código del programa de DHCP, en un impulso me puse a jugar y curiosear un poco. Aún cuando el cliente de DHCP no era tan extendible como el que nosotros ofrecemos con NT5 (por ejemplo, no buscaba opciones arbitrarias ni almacenaba los resultados), era obvio cómo podía escribir el código adicional para implementar esas funcionalidades. El programa completo consistía de cerca de 2600 líneas de código.

Un ejemplo de funcionalidad esotérica y extendida que era claramente un parche añadido por un tercero era un conjunto de rutinas que ensamblaban las solicitudes DHCP con cadenas específicas requeridas por sitios de Modem por cable y ADSL.

Algunos pocos pasos adicionales fueron necesarios para configurar el cliente DHCP para iniciar y configurar automáticamente mi tarjeta de red al arrancar la máquina, pero estos estaban documentados en el código del programa y en la documentación del desarrollador Húngaro.

Mis habilidades como programador UNIX son pobres pero fue inmediatamente obvio para mí como extender incrementalmente el código del cliente DHCP (la sensación era exilarante y adictiva).

Adicionalmente, debido directamente a la GPL + el hecho de tener el ambiente de desarrollo completo en frente de mi, estaba en posibilidades de escribir mis cambios y enviarlos por email en un par de horas (en contraste de cómo se llevarían a cabo cosas como estas en NT). Involucrarme en ese proceso me prepararía para un proyecto de linux más grande y ambicioso en el futuro[29].

"La sensación era exhilarante y adictiva". Paren las máquinas: Microsoft verifica experimentalmente el corolario metafórico de Moglen a la ley de Faraday. Envuelve con la internet cada cerebro en el planeta y gira al planeta. El software fluye en los cables. Crear es una propiedad emergente de las mentes humanas. "Debido directamente a la GPL", como Vallopillil acertadamente señaló, el software libre puso a su disposición un incremento exhilarante a su propia creatividad, de un tipo inalcanzable en su trabajo diurno para la mayor compañía de programación en el mundo. Si tan sólo hubiera mandado por email esa primera reparación adictiva, quien sabe donde estaría hoy en día.

29. Vinod Vallopillil, Análisis competitivo del OS Linux (Halloween II). Noten la sorpresa de Vallopillil porque un programa escrito en California fue subsecuentemente documentado por un programador en Hungría.

Así, al final, mis enanos amigos, es solo una cosa humana. Más bien como el por qué canta Fígaro, el por qué escribió Mozart la música para que la cantara, y el por qué inventamos todos palabras nuevas: Porque podemos. Homo ludens, conozcan al Homo faber. La condición social de interconexión global que llamamos la internet hace posible para todos nosotros el ser creativos en modos nuevos y previamente inimaginables. A menos que permitamos que la "propiedad" interfiera. Repitan conmigo, uds enanos y hombres: ¡Resistan la resistencia!

IV. ¿Mueren sus señorías en la oscuridad?

Para el androideIP, recién bajado del avión tras una semana en el Bellagio pagado por Dreamworks SKG, es suficiente para ocasionarle indigestión.

¿Liberar las posibilidades de la creatividad humana conectando a todos con todos los demás? ¿Quitar del camino al sistema de propiedad privada para que podamos todos añadir nuestras voces al coro, incluso si eso significara pegar nuestro canto sobre el tabernáculo mormón y mandar el resultado a un amigo? ¿Con Nadie sentado boquiabierto frente a la mezcla televisiva de violencia e inminente cópula cuidadosamente diseñadas para incrementar el interés de los ojos del joven hombre en un anuncio de cerveza? ¿Que será de la civilización? ¿O al menos de los profesores de copyright?

Pero tal vez esto sea prematuro. He estado hablando sólo de software. Software real, del tipo viejo, del que hace funcionar computadoras. No como el software que hace funcionar reproductores de DVD, o del tipo hecho por The Grateful Dead. Oh sí, los Grateful Dead. Había algo raro con ellos, ¿o no?, No prohibían que se grabara en sus conciertos. No les importaba si sus fans irritaban a la industria disquera. Parecen haberla hecho bien, sin embargo, deben admitirlo. El senador Patrick Leahi, ¿No es un ex fan de los Grateful Dead? Me pregunto si votará para extender los términos de la autoría corporativa a 125 años para que Disney no pierda a su ratón en 2004. Y esos reproductores de DVD - son computadoras, ¿no es así?

En la sociedad digital, todo está conectado. No podemos depender a la larga de distinguir una cadena de bits de otra para saber qué reglas aplicar. Lo que le pasó al software ya le está pasando tambíen a la música. Los señoríos de la industria disquera están batallando salvajemente para retener el control de la distribución, al tiempo que tanto los músicos como los espectadores se dan cuenta de que la gente de enmedio ya no es necesaria. El gran pueblo Potemkin de 1999, la llamada Iniciativa de la música digital segura, se habrá colapsado mucho antes de que el primer presidente de internet empiece su mandato, por simples razones técnicas tan obvias para los que sabemos como aquellas que dictaron el triunfo del software libre [30]. La revolución anarquista en la música es diferente de la del software sin más. pero ahí también --como cualquier adolescente con una colección de MP3 de música lanzada directamente por artistas independientes puede decir-- la teoría ha sido liquidada por los hechos. Ya seas Mick Yagger, o un gran artista nacional del tercer mundo buscando una audiencia global, o un improvisado de sótano que está reinventando la música, la industria disquera pronto no tendrá nada que ofrecerte que no puedas obtener mejor de forma gratuita. Y la música no suena peor cuando se distribuye gratuitamente, págale lo que quieras directamente al artista, y no pagues nada si no quieres. Dáselo a tus amigos; podría gustarles.

30. Ver "Están tocando nuestra canción: el día que la industria disquera murió", en el próximo a salir: La barbacoa invisible.

Lo que le pasó a la música también le está pasando a las noticias. Las agencias de información, como cualquier estudiante de leyes aprende incluso antes de tomar el curso casi obligatorio de copyright para androides, tienen un interés de propiedad protegible en sus modos de expresar las noticias, aunque no en los hechos que reportan

las mismas [31]. ¿Entonces porqué están regalando toda su producción? Porque en el mundo de la Red la mayoría de las noticias son bienes de consumo. Y la ventaja original de los recopiladores de noticias, de que estaban conectados internamente de modos en que otros no, cuando las comunicaciones eran caras, ha desaparecido. Ahora lo que importa es cosechar ojos para entregárselos a los anunciantes. No son las agencias de información las que tienen la ventaja al cubrir los eventos en Kosovo, eso es seguro. Mucho menos esos ejemplos de propiedad "intelectual", los señoríos televisivos. Ellos, con su excesivamente pagada gente bonita y su infraestructura técnica masiva, son de las únicas organizaciones en el mundo que no pueden permitirse estar en todos lados todo el tiempo. Además deben limitarse a noventa segundos por noticia, o los cazadores de ojos irán a otro lado. Así pues, ¿quién hace mejores noticias, los propietarios o los anarquistas? Lo veremos pronto.

Oscar Wilde dice en algún lugar que el problema con el socialismo es que necesita demasiadas tardes. Los problemas con el anarquismo como sistema social también son acerca de los costos de las transacciones. Pero la revolución digital altera dos aspectos de la economía política que de otro modo no habían variado a lo largo de la historia de la humanidad. Todo el software tiene cero costos marginales en el mundo de la Red, al mismo tiempo, los costos de la coordinación social se han reducido lo suficiente como para permitir la rápida formación y disolución de agrupaciones sociales muy diversas y de gran escala y todo sin limitaciones geográficas [32]. Tal cambio fundamental en las circunstancias materiales de la vida produce necesariamente del mismo modo cambios fundamentales en la cultura. ¿No lo crees? Dícelo a los Iorquois. Y por supuesto, tales profundos deslizamientos en la cultura son amenazas a las relaciones de poder existentes. ¿No lo crees? Pregúntale al partido comunista chino. O espera 25 años y ve si puedes encontrarlos para encuestarlos.

31. Agencia de información internacional Vs. Associated Press, 248 U.S. 215 (1918). Respecto de las tersas y meramente funcionales expresiones de las noticias emergentes que de hecho están en juego en los empellones entre las agencias de información, esta fue siempre una distinción que sólo un androide podía amar.

32. Ver "Sin hijo pródigo: La teoría política de la interconexión universal," ne el próximo The invisible Barbecue.

En este contexto, la obsolescencia del androideIp no es ni impredecible ni trágica. De hecho podría encontrarse a sí mismo retumbando estruendosamente en el desierto, aún explicando lúcidamente a una audiencia imaginaria las lucrativamente complicadas reglas de un mundo que ya no existe. Pero al menos tendrá compañía familiar, reconocible por todos esos relucientes partidos en Davos, Hollywood, y Bruselas. Los Amos de los medios están encontrándose con el destino, a pesar de cuanto sientan ellos que la Fuerza está de su lado. Las reglas sobre las cadenas de bits son ahora de dudosa utilidad para mantener el poder apropiándose de la creatividad humana. Vistos claramente a la luz de los hechos, estos emperadores tienen incluso menos ropa que las modelos que usan para capturar nuestros ojos. A menos que sean apoyados por tecnología que discapacite a los usuarios, una cultura de vigilancia penetrante que permita que cada lector de toda "propiedad" deba registrarse y pagar, y una pantalla de humo de aliento de androides asegurándole a cada joven que la creatividad humana se desvanecería sin la aristocracia benevolente de BillG el creador, de Lord Murdoch de

todos lados, el maestro de los juegos y el Señor gran ratón, su reinado está casi acabado. Pero lo que está en juego es el control del recurso más escaso de todos: nuestra atención. Reclutar es lo que hace todo el dinero del mundo en la economía digital, y los actuales dueños de la tierra pelearán por ello. Unidos contra ellos están solamente los anarquistas: dones nadies, hippies, aficionados, amantes, y artistas. El desigual enfrentamiento resultante es el gran asunto político y legal de nuestro tiempo. La aristocracia se ve difícil de vencer, pero así era como se veía en 1788 y 1913 también. Es, como Chou En-Lai decía sobre el significado de la revolución francesa, demasiado pronto para saber.

Sobre el autor

Eben Moglen es un profesor de Leyes e Historia Legal, Columbia Law School.
E-mail: moglen@columbia.edu

Reconocimientos.

Este ensayo fue preparado para una lectura en la Conferencia internacional Buchmann sobre leyes, tecnología e información, en la universidad de Tel Aviv, Mayo de 1999; Le agradezco a los organizadores por su amable invitación. Estoy en deuda como siempre con Pamela Karlan por su visión e impulso. Deseo agradecer especialmente a los programadores al rededor del mundo que hicieron posible al software libre.

Notas

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.