

# **er Anarchismus triumphiert**

von Eben Moglen 

Die Verbreitung des Linux Betriebssystems hat der gesamten Free-Software-Bewegung zu mehr Aufmerksamkeit verholfen. Dieses Papier zeigt, warum freie Software, die alles andere als ein Schattendasein neben dem kommerziellen Software-Markt führt, der erste nachhaltige Schritt zum Ableben des Schutzsystems geistigen Eigentums ist.

## **Inhalt**

**I. Software als Eigentum: Das theoretische Paradox**

**II. Software als Eigentum: Das praktische Problem**

**III. Anarchismus als eine Produktionsform**

**IV. Ihre Lordschaften sterben im Dunkeln ?**

## **Fazit**

### **I. Software als Eigentum: Das theoretische Paradox**

SOFTWARE: Kein anderes Wort hat so gründlich mit den praktischen und sozialen Effekten der digitalen Revolution zu tun. Ursprünglich handelte es sich um einen rein technischen Begriff. Er beschrieb die Teile eines Computersystems, die, anders als die unveränderlich als Systemelektronik hergestellte "Hardware", frei verändert werden konnten. Die erste Software beschränkte sich auf Steckverbindungen von Kabeln oder Schaltern an der Außenwand eines elektronischen Gerätes, aber sobald sprachliche Mittel entwickelt wurden, das Verhalten von Computern zu steuern, wurden unter "Software" Ausdrücke in für den Menschen mehr oder weniger lesbarer Sprache verstanden, die Maschinenverhalten sowohl beschrieben als auch kontrollierten [\[1\]](#).

Jenes war damals und dieses ist jetzt. Technologie, die auf der Verarbeitung von digital "codierter" Information basiert, dominiert heute die meisten Aspekte menschlicher Kultur in den "entwickelten" Gesellschaften [\[2\]](#). Der Wechsel von analoger zu digitaler Darstellung - bei Videos, Musik, Druck, Telekommunikation und sogar in der Choreografie, der Religionsausübung und bei der sexuellen Befriedigung - macht potentiell alle Formen menschlicher, symbolischer Aktivität zu Software, zu veränderlichen Anweisungen für die Beschreibung und die Kontrolle maschinellen Verhaltens. Im Wege einer konzeptionellen Rückbildung, die typisch für das westliche wissenschaftliche Denken ist, wird der Gegensatz zwischen Hard- und Software heute in der natürlichen und sozialen Welt beobachtet und ist zu einer neuen Möglichkeit geworden, den Konflikt zwischen Determinismus und freier Willensentscheidung auszudrücken, zwischen Natur und Nahrung, zwischen Genen und Kultur. Unsere genetisch verdrahtete "Hardware" ist unsere

Natur, die uns bestimmt. Unsere Nahrung ist "Software", sie legt unsere kulturelle Programmierung an, in der unsere relative Freiheit liegt. Dies gilt in vielen Bereichen [3]. Und so wird "Software" zur universellen Metapher für alle symbolische Aktivität, offensichtlich losgelöst vom technischen Zusammenhang des Wortursprunges, auch wenn den technisch Versierten bei dieser Begriffsinflation unwohl wird, deren konzeptuelle Bedeutung oft übersehen wird [4].

Die weitverbreitete Annahme der Digitaltechnologie von denen, die ihre Funktionsprinzipien nicht verstehen, scheint zwar die umfängliche metaphorische Verwendung des Begriffs "Software" zu rechtfertigen, sie erlaubt uns aber nicht zu vergessen, daß Computer sich heute überall unter unserer sozialen Haut befinden. Der Wechsel von analog zu digital ist wichtiger für die Struktur unserer sozialen und rechtlichen Beziehungen als der berühmtere, aber weniger wahrscheinliche Wechsel von Status zu Vertrag [5]. Das sind schlechte Neuigkeiten für diejenigen Rechtswissenschaftler, die davon nichts verstehen - woraus sich auch erklärt, daß allenthalben fleißig vorgespielt wird, man verstünde etwas davon. Jedenfalls potentiell ist diese "große Überfahrt" aber eine sehr gute Neuigkeit für diejenigen, die solchermaßen neu gefundenes Land zu ihrem Eigentum machen können. Dies ist der Grund, warum die derzeitigen "Eigentümer" von Software die Ignoranz aller anderen so vehement unterstützen und fördern. Pech für sie - der Trick wird aus Gründen, mit denen auch Rechtsdenker Bekanntschaft machen mußten, denen die Anwendung ihrer traditionellen Logik in diesem Bereich mißglückt ist, nicht funktionieren. Dieser Artikel erklärt warum [6].

Zu Beginn müssen wir die uns die technischen Grundlagen der bekannten Geräte vergegenwärtigen, die uns in der Ära "kultureller Software" umgeben. Ein gutes Beispiel ist der CD-Player. Sein hauptsächlichlicher "Input" ist ein Datenstrom von einer optischen Speicherplatte. Der Datenstrom beschreibt Musik in Form von Einheiten, etwa 44.000 Mal pro Sekunde, von Frequenz und Amplitude jeweils in zwei Audio-Kanälen. Sein hauptsächlichlicher "Output" sind analoge Audio-Signale [7]. Wie alles in der digitalen Welt ist Musik, aus der Sicht des CD-Players, bloße numerische Information. Eine bestimmte Aufnahme von Beethovens Neunter mit Arturo Toscanini und dem NBC Symphonie-Orchester und Chor ist (um nur einige unbedeutende Nummern zu verwenden) 1276749873424, während Glenn Goulds besonders bemerkenswerte letzte Aufnahme der Goldberg Variationen (ebenso vereinfacht) 767459083268 ist.

Merkwürdigerweise sind diese zwei Zahlen "urheberrechtlich geschützt". Das heißt angeblich, daß niemand eine weitere Kopie dieser Zahlen besitzen kann, nachdem sie einmal in physischer Form fixiert sind, ohne Inhaber eines Verwertungsrechts zu sein. Und niemand kann aus 767459083268 für seine Freunde 2347895697 machen (etwa um Goulds lächerliches Verständnis von Tempi zu korrigieren), ohne eine "Werkveränderung" vorzunehmen, die der Lizenzierung bedarf.

Gleichzeitig beinhaltet eine andere optische Speicherplatte eine andere Nummer, etwa 7537489532. Diese ist ein Algorithmus für die lineare Programmierung großer Systeme mit verschiedenen Regelgrößen, der zum Beispiel für die optimale Ausnutzung der Güterwaggon-Kapazitäten eines Transportunternehmens nützlich ist. In den USA ist diese

Nummer "patentiert", was bedeutet, daß niemand 7537489532 selbst herleiten darf, oder auf andere Weise "die Kunst" des Patentes zur Lösung von Problemen der linearen Programmierung "ausüben" darf. Dabei spielt es keine Rolle, wie er an diese Nummer gekommen hat, sie gegebenenfalls gar selbst herausgefunden hat. Er benötigt eine Lizenz des Nummerneigentümers.

Dann ist da 9892454959483. Dies ist der Programmtext (Source-Code) von Microsoft Word. Dieser ist nicht nur urheberrechtlich geschützt, sondern zudem ein "trade secret", ein Geschäftsgeheimnis. Das heißt, wer diese Nummer Microsoft wegnimmt und sie jemand anderem zuspielt, macht sich strafbar.

Zuletzt ist da 588832161316. Diese Nummer tut nichts, sie ist nur die Quadratzahl von 767354. Soweit ich weiß, gehört sie noch niemandem unter einer der genannten Kategorien. Noch nicht.

An dieser Stelle müssen wir uns mit dem ersten Einwand der Gelehrten auseinandersetzen. Er kommt von einer als IP-Druide bekannten Kreatur. Der IP-Druide hat einen entwickelten Geist und lebt ein kultiviertes Leben. Er schätzt sehr die eleganten Dinners bei akademischen und ministeriellen Konferenzen über das TRIPS, ganz zu schweigen von den zahlreichen Auftritten bei MSNBC. Er möchte anmerken, daß hier der Fehler gemacht wird, die Verkörperung mit dem geistigen Eigentum selbst zu verwechseln. "Es ist doch nicht die Nummer, die patentiert wird, Dummkopf, sondern nur der Kammerkar-Algorithmus". Die Nummer *könne* geschützt werden, weil das Urheberrecht die expressiven Eigenschaften einer bestimmen körperlichen Darstellung einer Idee umfasse, nicht aber den Algorithmus. Wogegen die Nummer wiederum nicht patentierbar sei, sondern nur das "Lehren" der Nummer mit dem Ziel, die Güterwaggons pünktlich fahren zu lassen. Und die Zahl schließlich, die den Source-Code von Microsoft Word darstelle, könne ein Geschäftsgeheimnis sein, aber wenn jemand sie selbst herausfände (etwa im Wege arithmetischer Manipulation anderer, von Microsoft veröffentlichter Zahlen, bekannt als "reverse engineering"), werde er nicht bestraft - jedenfalls wenn er in bestimmten Teilen der Vereinigten Staaten lebe.

Der Druide hat, wie andere Druiden auch, häufig recht. Die Voraussetzung für ein Dasein als Druide ist, alles über etwas Bestimmtes zu wissen und nichts über den Rest. Durch seine zeitige und dringende Intervention hat der Druide klargestellt, daß das derzeitige Urheberrechtssystem zahlreiche Kniffe und Tücken enthält. Diese Komplexitäten erlauben es Professoren meinungsüberlegen zu sein, Abgeordneten, Wahlkampfspenden zu bekommen, Rechtsanwälten, schöne Anzüge zu tragen, und sie erlauben es Murdoch, reich zu sein. Sie haben sich zum größten Teil in der Zeit industrieller Informationsverbreitung entwickelt, als Information in analoger Form auf physischen Träger gespeichert wurde, deren Herstellung, Transport und Vermarktung erhebliche Kosten verursachte. Werden diese Komplexitäten auf digitale Informationen angewandt, die sich bruchlos durch das Netzwerk bewegen und keinerlei Grenzkosten pro Kopie verursachen, machen sie nur noch für den halbwegs Sinn, der nicht zu schielen aufhört.

Aber dies ist nicht das Anliegen dieses Artikels. Worum es geht ist, daß unsere Welt zunehmend aus nichts anderem als großen Zahlen (auch als Datenströme bezeichnet)

besteht und daß unser Rechtssystem derzeit aus Gründen, die nichts mit dem sich entwickelnden Eigentum an den Nummern selbst zu tun haben, einander ähnliche große Zahlen vollkommen unterschiedlich behandelt. Niemand kann nach einem bloßen Blick auf eine Zahl, die etwa 100 Millionen Ziffern lang ist, sagen, ob diese Nummer patentiert, urheberrechtlich oder als Handelsgeheimnis geschützt ist, oder überhaupt irgendjemandem "gehört". Das Rechtssystem behandelt also - Gott sei es gedankt, wenn wir Urheberrechtsprofessoren, Abgeordnete, Gucci-Träger oder Der Große Rupert höchstselbst sind - nicht unterscheidbare Dinge auf unterschiedliche Art und Weise.

In meiner Rolle als Rechtshistoriker, der mit der säkularen (d.h. sehr langfristigen) Entwicklung rechtlichen Denkens befaßt ist, behaupte ich, daß rechtliche Regime, die auf scharfen, aber unvorhersehbaren Unterscheidungen wesentlich gleicher Objekte aufbauen, extrem instabil sind. Sie fallen mit der Zeit auseinander, weil jeder Anwendungsfall eine Einladung an mindestens eine Partei darstellt, die Zuordnung des streitigen Objekts nicht in Kategorie A, sondern in die den Interessen des Beschwerdeführers stärker entgegenkommende Kategorie B zu verlangen. Diese Spiele - ob eine Schreibmaschine zum Zweck günstiger Frachttarife als Musikinstrument gilt oder ein Dampf-Bagger als Motorfahrzeug - sind der Stoff juristischer Geschicklichkeit. Aber wenn mit einem Mal konventionell angewandte rechtliche Kategorien vom Richter verlangen, zwischen dem Identischen zu unterscheiden, wird das Spiel unendlich langwierig, unendlich teuer und für den unvoreingenommenen Betrachter nahezu unendlich abstoßend [8].

Daher können die Parteien soviel Geld ausgeben wie sie wollen, für so viele Abgeordnete und Richter wie sie sich leisten können - und das sind eine ganze Menge für die neuen "Eigentümer" der digitalen Welt - aber die Regeln, die sie kaufen, werden am Ende nicht funktionieren. Früher oder später werden ihre Paradigmen zusammenbrechen. Natürlich, wenn "später" heißt: zwei Generationen von heute an, wird die bis dahin erfolgte Verteilung von Reichtum und Macht durch kaum eine mildere Maßnahme rückgängig zu machen sein, als durch einen *bellum servile* zwischen Couch-Potatoes und Medienmagnaten. Es ist also nicht genug zu wissen, daß die Geschichte nicht auf Bill Gates Seite ist. Wir sagen die Zukunft in sehr beschränktem Umfang voraus: Wir wissen, daß die existierenden Regeln, die heute noch die ganze Stärke konventioneller Überzeugung hinter sich wissen können, nicht länger Bedeutung haben. Betroffene werden sie nach Belieben gebrauchen und mißbrauchen bis die "respektierte" konservative Meinungsmehrheit ihr Ende anerkennt - mit ungewissem Ergebnis. Aber die realistisch denkende Lehre sollte ihre Aufmerksamkeit bereits dem dringenden Bedürfnis nach neuen Denkmustern zuwenden.

Wenn dieser Punkt erreicht ist, sehen wir uns mit dem anderen Hauptdarsteller gebildeter Idiotie konfrontiert: der Wirtschaftszwerg. Wie der IP-Druide ist der Wirtschaftszwerg eine Art Stachelschwein [9], aber wo der Druide sich von Logik statt von Erfahrung leiten läßt, liegt die Spezialität des Wirtschaftszwerges in einer engagiert vorgetragenen und auf den Punkt gebrachten, aber vollständig unzutreffenden Sicht der menschlichen Natur. Nach Ansicht des Wirtschaftszwerges ist der Mensch ein Wesen, das aus "Anreizen" besteht, die durch die Vorstellung des Kontostandes zu verschiedenen Zeitpunkten zum Leben erweckt werden. Deshalb fühlt sich der Wirtschaftszwerg zu dem Einwand berufen, ohne die Regeln, die hier für überflüssig gehalten werden, gäbe es keine Anreize, Dinge zu schaffen, die diese Regeln als Eigentum schützen: Ohne die Möglichkeit, andere von Musik

auszuschließen, gäbe es danach keine Musik, weil niemand sicher sein könnte, für ihre Schaffung bezahlt zu werden.

Musik ist nicht wirklich das Thema hier; die Software, um die es in diesem Beitrag geht, ist die alten Stils: Computerprogramme. Aber weil es dem Wirtschaftszweig darauf ankommt, wenigstens am Rand darüber zu sprechen und weil es, wie wir gesehen haben, nicht mehr wirklich möglich ist, Computerprogramme von Musikdarbietungen zu trennen, sollen ein oder zwei Worte dazu gesagt werden. Wenigstens haben wir so die *Genugtuung*, uns ein *argumentum ad pygmeam* zu gönnen. Bringt der Wirtschaftszweig es zu Reichtum bringt, geht er, meiner Erfahrung nach, in die Oper. Doch so oft er auch *Don Giovanni* hört, es kommt ihm nicht in den Sinn, daß - seiner Auffassung nach - Mozarts Schicksal Beethoven ein für allemal entmutigt haben müßte - oder, daß wir uns an der *Zauberflöte* erfreuen können, obwohl Mozart genau wußte, daß er dafür nicht bezahlt werden würde. Tatsächlich sind die *Zauberflöte*, die *Matthäuspassion* und die Motetten des Frauenmörders Carlo Gesualdo alle Teil einer jahrhundertelangen Tradition von freier Software in einem weiteren Sinne. Dies vermag der Wirtschaftszweig nicht anzuerkennen.

Das Grundproblem des Zwergen liegt darin, daß der Begriff "Anreize" bloß eine Metapher ist, und als Metapher zur Beschreibung kreativer menschlicher Aktivität ist es eine ziemlich unbefriedigende. Ich habe das bereits früher gesagt [10], aber die bessere Metapher entstand an dem Tag, als Michael Faraday erstmals bemerkt hat was passiert, wenn er eine Spule Draht um einen Magneten herum wickelt und diesen dann in Rotation versetzt. In einem solchen Draht entsteht Spannung, aber wir fragen nicht, was der "Anreiz" für die Elektronen ist, ihre Behausung zu verlassen. Wir sagen, daß die Spannung aus einer Eigenschaft dieses Systems resultiert, die wir Induktion nennen. Die Frage ist für uns: "Was ist der Widerstand des Drahtes?". Moglens Metaphorische Ableitung von Faradays Gesetz sagt also, daß wenn man das Internet um jedermann auf dem Planeten wickelt und den Planeten in Rotation versetzt, Software in dem Netzwerk fließt. Es ist eine Eigenschaft von vernetzten menschlichen Geistern, daß sie Dinge schaffen, um einander Freude zu machen und die Ungemütlichkeit des Alleinseins zu überwinden. Die einzige Frage ist: "Was ist der Widerstand des Netzwerkes?" Moglens Metaphorische Ableitung von Ohms Gesetz behauptet, daß der Widerstand des Netzwerkes direkt proportional zur Feldstärke des Systems "geistigen Eigentums" ist. Daher ist die richtige Antwort für den Wirtschaftszweig: Widerstehe dem Widerstand.

Natürlich klingt dies alles in der Theorie sehr schön. "Widerstehe dem Widerstand" klingt kraftvoll, aber wir hätten - ungeachtet aller Theorie - ein wirkliches Problem, wenn der Wirtschaftszweig Recht hätte und eine Unterproduktion guter Software entstünde, weil niemandem erlaubt wäre, sie zu besitzen. Aber Zwergen und Druiden sind nur Formalisten verschiedener Prägung und der Vorteil des Realismus liegt darin, daß er die Fakten auf seiner Seite hat, wenn er mit der Analyse der Fakten beginnt. Dabei stellt sich heraus, daß ein Eigentumsschutz für Software zu schlechter Software führt.

## II. Software als Eigentum: Das praktische Problem

Um zu verstehen, warum eine Umdeutung von Software in Eigentum zu schlechter Software führt, bedarf es einer Einführung in die Kunsthistorik. Tatsächlich sollte besser mit dem Wort "Kunst" selbst begonnen werden. Die Programmierung von Computern kombiniert zweckgerichtetes Denken mit literarischer Erfindungskunst.

Auf den ersten Blick scheint Source-Code eine nicht-literarische Kompositionsform zu sein [11]. Die Hauptanforderung an ein Computerprogramm ist, daß es läuft, das heißt, daß es sich entsprechend den Spezifikationen verhält, die seine Ausgaben in Abhängigkeit von den Eingaben beschreiben. Auf dieser generellen Ebene ist der Funktionsinhalt von Programmen alles, was sichtbar wird.

Funktionierende Computerprogramme jedoch sind Teil von Computersystemen, also von wechselwirkenden Ansammlungen von Hardware, Software und Menschen. Die menschliche Komponente eines Computersystems ist nicht nur sein Nutzer, sondern auch die (vermutlich verschiedenen) Personen, die das System warten und verbessern. Source-Code kommuniziert nicht nur - vermittelt durch den Compiler, der die maschinenverständliche Sprache produziert - mit dem Computer, der das Programm ausführt, sondern auch mit anderen Programmierern.

Die Funktion von Source-Code gegenüber anderen Menschen ist von Nicht-Programmierern nicht leicht zu verstehen, die Computerprogramme in erster Linie unverständlich finden. Sie wären überrascht zu wissen, daß der Großteil der in Computerprogrammen enthaltenen Informationen aus der Sicht des Compilers "Kommentar", also nicht-funktionelles Material ist. Die Kommentare sind natürlich an Dritte gerichtet, die gegebenenfalls ein Problem lösen oder die Funktionsweise des Programms ändern oder verbessern wollen. In den meisten Programmiersprachen wird viel mehr Platz darauf verwendet, anderen zu erklären, was das Programm gerade tut, als dem Computer zu erklären, was er tun soll.

Die Entwicklung von Programmiersprachen war immer von den zwei Anforderungen geprägt, zum einen die Anforderungen an die Maschine so gut wie möglich zu spezifizieren und zum anderen dem menschlichen Leser beschreibend zu informieren. Es lassen sich drei Grundstrategien erkennen, um diesen beiden Anforderungen gerecht zu werden. Der erste wird vor allem im Bereich maschinenspezifischer Programmiersprachen, der sogenannten Assembler, eingesetzt. Er sucht die an die Maschine und den Menschen gerichteten Kommandos zu separieren. Assembler-Kommandos sind mit Kommandos in Maschinensprache nahe verwandt: Eine Zeile des Programmtextes entspricht generell einem Kommando in Maschinensprache. Der Programmierer kontrolliert das Verhalten der Maschine so unmittelbar wie möglich. Wenn er diszipliniert ist, kommentiert er parallel und legt alle paar Hundert Zeilen eine Pause ein, um in "Blockkommentaren" die Funktionsweise des Programms zu erläutern oder die wesentlichen Datenstrukturen zu dokumentieren, mit denen das Programm gerade arbeitet.

Ein zweiter Ansatz, für den die Programmiersprache COBOL (für Common Business-Oriented Language) charakteristisch ist, versuchte, dem Programm selbst die Form von Anweisungen in natürlicher Sprache zugeben, die zwar umständlich, aber theoretisch vom

Menschen lesbar waren. Eine Zeile in COBOL könnte zum Beispiel lauten: "MULTIPLY PRICE TIMES QUANTITY GIVING EXPANSION." Als das Pentagon und Industrieexperten in den 60er Jahren mit der gemeinsamen Entwicklung an COBOL begannen, schien dies ein vielversprechender Ansatz zu sein. Weil COBOL-Programme weitgehend selbsterklärend schienen, konnten auch große Programme von gemischten Arbeitsgruppen entwickelt werden und konnten Programmierer ausgebildet werden, die, obwohl hochspezialisiert, keinen so tiefen Einblick in die Maschinensprache benötigten wie Assembler-Programmierer. Aber der Grad an Allgemeinheit, mit dem diese Programme sich selbst erklärten war falsch gewählt. Ein formelhafterer und komprimierterer Ausdruck des Programmschrittes "expansion = price x quantity" etwa, eignete sich besser sogar für Geschäfts- und Finanzanwendungen, deren Leser und Schreiber an mathematische Ausdrücke gewöhnt waren, zumal wegen der Umständlichkeit der die Programmdetails umschreibenden Sprache nicht darauf verzichtet werden konnte, Datenstrukturen und den Programmzusammenhang separat zu erklären.

Entsprechend begannen die Entwickler Ende der 60er Jahre, mit Programmiersprachen zu experimentieren, bei denen die Mischung aus detaillierten Anweisungen und für die Veränderung oder Reparatur benötigter Nebeninformation schwieriger wurde. Einige Programmierer gingen zu hochabstrakten und komprimierten Sprachen über, in denen Daten abstrakt verarbeitet wurden, so daß etwa "A x B" die Multiplikation zweier ganzer oder komplexer Zahlen, zweier Felder oder anderer Datentypen meinen konnte, die einem als "Multiplikation" bezeichneten Prozeß zugänglich waren, der vom Computer auf der Basis der Variableninhalte von A und B zu einem bestimmten Zeitpunkt ausgeführt werden sollte [12]. Weil so die Programme extrem knapp ausfielen, war man der Meinung, das Verständnisproblem für diejenigen, die später Veränderungen oder Reparaturen am Programm durchführen wollten, vereinfacht zu haben. Indem die technischen Details der eigentlichen Computerfunktion verborgen und der Algorithmus betont wurde, bildeten sich Sprachen heraus, die besser als Englisch oder jede andere natürliche Sprache geeignet waren, schrittweise Prozesse auszudrücken. Kommentar wäre dort nicht nur unnötig, sondern sogar störend, genau so wie gebräuchliche Metaphern, die mathematische Zusammenhänge in Englisch erläutern sollen, mehr verwirren als erhellen.

### **Wie wir den "Microbrain" - Ärger erschufen**

Die Geschichte der Programmiersprachen spiegelt direkt das Bedürfnis wider, Formen von Kommunikation zwischen Mensch und Maschine zu finden, die es auch ermöglichen, komplexe Konzepte für den menschlichen Leser verständlich zu machen. "Deutlichkeit" wurde nicht deshalb eine Haupteigenschaft von Programmiersprachen, weil sie den Programmablauf vereinfacht hätte, sondern weil sie die gemeinschaftliche Schaffung und Wartung komplexer Software-Systeme vereinfacht.

Auf den ersten Blick scheint dies die Anwendung traditionellen "Copyright"-Denkens auf die entstehenden Werke zu rechtfertigen. Obwohl hauptsächlich aus "funktionellen" Elementen bestehend, beinhalten Computerprogramme "expressive" Bestandteile von erheblicher Wichtigkeit. Das Urheberrecht erkennt den Zusammenschluß von Funktion und Ausdruck als charakteristisch für viele Arten urheberrechtlich geschützter Werke an. Also war "Source-Code", der sowohl die für die Funktion der Maschine nötigen

Informationen als auch den für den menschlichen Leser gedachten "Kommentar" enthielt, ein idealer Kandidat für das Urheberrecht.

Das ist solange wahr, wie Klarheit darüber besteht, daß die expressive Komponente ausschließlich zu dem Zweck existierte, die Anfertigung von "Werkveränderungen" durchzuführen. Wäre es nicht um Änderungen zu erleichtern, wären alle expressiven Elemente von Software völlig überflüssig und Source-Code wäre genausowenig urheberrechtlich zu schützen wie reine Maschinensprache, die von allen nichtfunktionellen Bestandteilen befreit ist.

Der Zustand der Computerindustrie in den 60er und 70er Jahren, als die Fundamente anspruchsvoller Computerprogrammierung gelegt wurden, verdeckte die Spannung, die in dieser Situation angelegt ist. Zu jener Zeit war Hardware teuer. Computer waren zunehmend große und komplexe Ansammlungen von Maschinen und das Geschäft, diese Maschinenmengen für den allgemeinen Gebrauch zu entwickeln und zu bauen wurde dominiert, wenn nicht monopolisiert von einem Unternehmen. IBM verschenkte seine Software. Natürlich besaß das Unternehmen die von seinen Mitarbeitern geschriebenen Programme und ließ deren Text urheberrechtlich schützen. Aber ebenso lieferte es seinen Kunden die Programme - inklusive des Source-Codes - ohne Aufpreis mit und ermutigte sie, daran Verbesserungen und Anpassungen vorzunehmen und mitzuteilen. Für einen dominanten Hardware-Produzenten ging diese Strategie auf: Bessere Programme halfen, mehr Computer zu verkaufen und darin lag die Profitabilität des Geschäfts.

In dieser Zeit waren Computer auf die Bedürfnisse bestimmter Organisationen zugeschnitten aber nicht darauf, auf breiter Front miteinander zu kommunizieren. Die benötigte Software wurde nicht über Netzwerke vertrieben, sondern auf Magnetbändern. Dieses Verbreitungssystem führte zu einer Zentralisierung der Softwareentwicklung, so daß die erlaubten Veränderungen und Verbesserungen durch IBM-Kunden in erster Linie dem Unternehmen IBM zugute kamen, das dann entschied, ob und wie sie Aufnahme in die zentral entwickelte und vermarktete Software-Version fanden. In zweierlei wichtiger Hinsicht also war die beste Computer-Software der Welt frei: Sie war kostenlos anzuschaffen und ihre Lieferbedingungen erlaubten und förderten Experimentieren, Änderungen und Verbesserungen [13]. Daß die in Frage stehende Software unter dem anwendbaren Urheberrecht IBMs Eigentum war, brachte zwar einige Einschränkungen für die Nutzer mit sich, ihre Verbesserungen und Anpassungen selbst zu vermarkten, aber in der Praxis wurde Mainframe-Software kooperativ von den Hardware-Herstellern und ihren technisch versierten Nutzern geschaffen, wobei die Vertriebsmöglichkeiten des Herstellers genutzt wurden, um die erreichten Verbesserungen in der Nutzergemeinschaft durchzusetzen. Das Recht, andere auszuschließen, eines der wichtigsten "Scheite im Bündel" der geistigen Eigentumsrechte (ein vom US Supreme Court gern verwendetes Bild), war für den Kern des Software-Geschäfts praktisch bedeutungslos, ja sogar unerwünscht [14].

Nach 1980 wurde alles anders. Innerhalb von zehn Jahren machte die Welt der Mainframe-Rechner Platz für die Welt der Heim-PCs. Und als ein "Erfolg" der Industrieentwicklung wurde das mit Abstand wichtigste Element der Software für Heimcomputer, das Betriebssystem, das einzig nennenswerte Produkt eines Unternehmens,

das keine Hardware herstellte. Basissoftware von hoher Qualität war nicht länger ein Teil der Produktstrategie von Hardware-Produzenten. Statt dessen bestimmte ein Unternehmen mit überwältigendem Marktanteil und mit dem einem Quasi-Monopolisten eigenen Desinteresse an der Förderung von Vielfalt die Standards der Softwareindustrie. In diesem Zusammenhang erhielt das Recht, andere von der Beteiligung an der Produktentwicklung auszuschließen, entscheidende Bedeutung. Microsofts Marktmacht beruhte ausschließlich auf seinem Eigentum am Source-Code des Windows-Betriebssystems.

Für Microsoft waren andere, die "Werkveränderungen" herstellten, was sonst als Reparaturen und Verbesserungen verstanden wurde, eine Bedrohung seiner Geschäftsgrundlage. In der Tat bestand Microsofts Strategie - wie inzwischen auch mehrere Gerichtsentscheidungen angedeutet haben - darin, innovative Ideen irgendwo auf dem Software-Markt aufzuspüren, aufzukaufen und sie dann entweder von der Bildfläche verschwinden zu lassen oder in das eigene Produkt aufzunehmen. Die Kontrolle über die Grundfunktionen von Computern zu behalten, die von anderen hergestellt, verkauft, besessen und benutzt wurden, gab Microsoft einzigartige und profitable Möglichkeiten, die gesamte PC-Kultur zu beeinflussen [15].

Soweit es um die Qualität von Software geht, war das Ergebnis niederschmetternd. Das Monopol war ein reiches und mächtiges Unternehmen, das eine Vielzahl von Programmierern beschäftigte, aber sich nicht die Zahl von Testern, Designern und Entwicklern leisten konnte, die zur Produktion solcher Software nötig gewesen wäre, die unter den verschiedensten Bedingungen, denen zunehmend ubiquitär eingesetzte PCs ausgesetzt waren, flexibel, robust und technisch innovativ arbeitet. Microsofts Marketing-Strategie bestand darin, sein Produkt auf die technisch ahnungslosesten Nutzer zuzuschneiden und "Angst, Unsicherheit und Zweifel" (engl. "fear, uncertainty, and doubt", innerhalb Microsofts als "FUD" bekannt) zu benutzen, um technisch Versierte von der Benutzung potentieller Konkurrenzprodukte abzuhalten, deren langfristige Verfügbarkeit angesichts der Marktmacht von Microsoft immer ungewiß war.

Ohne kontinuierliche Wechselwirkungen zwischen den Nutzern, die zur Reparatur und Verbesserung in der Lage waren und dem Betriebssystemhersteller konnte der unweigerliche Qualitätsverfall nicht gestoppt werden. Weil aber die PC-Revolution die Zahl der Computernutzer exponentiell wachsen ließ, hatten diejenigen, die mit dem etablierten System in Kontakt kamen nichts, womit sie es hätten vergleichen können. Ohne die Stabilitäts- und Qualitätsstandards, die Wartungsfreundlichkeit und Effizienz je gekannt zu haben, die in der Mainframe-Welt gang und gäbe gewesen waren, konnte von PC-Nutzern kaum verlangt werden zu begreifen, wie schlecht - relativ gesehen - die Produkte des Software-Monopols funktionierten. Als Leistung und Kapazität der Rechner wuchsen, wurden die Defizite im allgemeinen Leistungsanstieg weniger deutlich. Normalnutzern, von der Angst vor einer Technologie gezeichnet, die für sie fast gänzlich unverständlich war, kam die Fehlerhaftigkeit des Betriebssystems sogar ganz recht. In einer Wirtschaft, die sich in einem mysteriösen Wandel befindet, der Millionen von Karrieren zu zerstören droht, war es auf eine perverse Weise beruhigend, daß kein PC in der Lage zu sein schien, mehr als ein paar Stunden ohne Absturz zu arbeiten. Obwohl es frustrierend war, mit jedem unnötigen Rechnerproblem gerade erledigte Arbeit zu verlieren, hatte die

offensichtliche Fehlbarkeit von Computern etwas Beruhigendes [16].

Nichts davon wäre nötig gewesen. Die niedrige Qualität von PC-Software hätte erhöht werden können, wenn man die Nutzer in den natürlicher Weise evolutionären Prozeß von Software-Design und -Implementierung mit eingebunden hätte. Eine Lamarcksche Herangehensweise, in der Verbesserungen überall und von jedermann hätten vorgenommen werden können, die von jedem hätten übernommen werden können, hätte das Defizit überwunden und in der PC-Welt die Stabilität und Verlässlichkeit hergestellt, die im quasi-proprietären Umfeld der Mainframe-Ära selbstverständlich war. Aber Microsofts Geschäftsmodell schloß die Lamarcksche Vererbung von Software-Fortschritten aus. Die Urheberrechtsdogmatik, im Allgemeinen und im Hinblick auf Software im Besonderen, führt zu einer Überbetonung des Schöpferischen. Das Problem dabei ist, daß Bill G., der Schöpfer, weit davon entfernt ist, unfehlbar zu sein, und tatsächlich hat er das auch gar nicht versucht.

Wie um die Ironie noch deutlicher werden zu lassen, macht das Wachstum des Netzwerks die nicht-proprietäre Alternative noch praktikabler. Was akademisches und populäres Schrifttum einmütig als ein Ding ("das Internet") bezeichnen, ist tatsächlich der Name für eine bestimmte Form sozialen Zusammenlebens, nämlich, daß jeder in der Netz-Gesellschaft direkt und ohne Vermittlung mit jedem anderen verbunden ist [17]. Der weltweite Zusammenschluß von Netzwerken hat die Engstelle beseitigt, deretwegen in der Mainframe-Ära ein zentraler Software-Hersteller benötigt wurde, um die Ergebnisse individueller Innovation zu rationalisieren und zu verbreiten.

Letztlich, als kleine Ironie der Geschichte, wurde der Triumphzug schlechter Software im PC-Zeitalter durch eine überraschende Bündelung von Kräften umgekehrt: der durch das Netz eingeleitete soziale Wandel, eine lange vergessene europäische politische Theorie und eine kleine Gruppe von Programmierern auf der ganzen Welt, die von einer einzigen einfachen Idee angetrieben wurden.

### **Software will frei sein, oder: Wie wir aufhörten, uns Sorgen zu machen und die Bombe zu lieben lernten**

Lange bevor das Netz der Netze eine praktische Realität geworden war, sogar bevor es am Horizont auftauchte, gab es den Wunsch, daß Computer mit Software arbeiten, die jedermann frei zur Verfügung steht. Es begann als eine Reaktion gegen proprietäre Software in der Mainframe-Ära und verlangt wiederum nach einem kurzen historischen Abriß.

Obwohl IBM der größte Verkäufer von vielseitig einsetzbaren Computern in der Mainframe-Ära war, war es nicht der größte Designer und Hersteller solcher Hardware. Der Telefonmonopolist, American Telephone & Telegraph (AT&T) war in Wirklichkeit größer als IBM, setzte aber seine Produkte intern ein. Und bei der berühmten Forschungstochter des Telefonmonopols, Bell Labs, entstand in den späten 60er Jahren im Rahmen der oben beschriebenen Entwicklung von Computersprachen das Unix-Betriebssystem.

Die Idee von Unix war es, ein einziges Betriebssystem zu schaffen, das auf die Größe aller verschiedenen Computer zugeschnitten werden konnte, die die Telefongesellschaft für sich selbst herstellte. Um dieses Ziel zu erreichen, konnte das Betriebssystem weder in Maschinensprache noch in Assembler geschrieben werden, dessen linguistische Form immer von der betreffenden Hardwareform abhing, sondern in einer deutlicheren und allgemeineren Sprache. Diejenige, die dafür ausgewählt wurde, war ebenfalls eine Bell Labs-Entwicklung, genannt "C" [18]. Die C-Sprache wurde für vielfältige Programmieraufgaben üblich, ja sogar dominant, und in den späten 70er Jahren war das Unix-Betriebssystem schon auf Computer vieler Hersteller und unterschiedlichen Zuschnitts übertragen (im Fachjargon "portiert") worden.

AT&T vertrieb Unix intensiv und dies mußte wegen des Designs dieses Betriebssystems im C-Source-Code geschehen. AT&T behielt aber das Eigentum an "C" und zwang die Nutzer, Lizenzen zu erwerben, die den Weitervertrieb und die Anfertigungen von Werkveränderungen verboten. Große industrielle oder akademische Rechenzentren konnten sich solche Lizenzen leisten, aber der Einzelne nicht, während gleichzeitig die Lizenzbestimmungen Unix benutzende Programmierer daran hinderten, das System kontinuierlich und nicht nur episodisch zu verbessern. Und als weltweit Programmierer die PC-Revolution nicht nur ahnten, sondern sogar erwarteten, herrschte Betroffenheit über den "unfreien" Status von Unix.

Zwischen 1981 und 1984 begann ein Mann einen Kreuzzug, um diese Situation zu verändern. Richard M. Stallman, zu dieser Zeit Angestellter im Labor für künstliche Intelligenz des Massachusetts Institute of Technology (MIT Artificial Intelligence Laboratory), plante das Projekt der unabhängigen und kooperativen Konzeption und Einführung eines Betriebssystems, das wirklich freie Software sein sollte. In Stallmans Worten sollte freie Software eine Frage der Freiheit sein, nicht des Preises. Jeder sollte solche Software frei verändern und neu vertreiben oder verkaufen können mit der einzigen Einschränkung, daß er nicht versuche, die Rechte seiner Abnehmer zu beschneiden. So konnte freie Software ein selbstlaufendes Projekt werden, in dem keine Innovation durch die proprietäre Rechteaübung verlorenginge. Das System, so entschied Stallman, sollte GNU genannt werden. GNU bedeutet (in erster Andeutung einer Vorliebe für rekursive Abkürzungen, die die freie-Software-Bewegung seitdem kennzeichnet) "GNUs Not Unix". Trotz Zweifeln hinsichtlich des fundamentalen Designs von Unix und seiner Vermarktungsart sollte GNU von der weiten, wenngleich "unfreien" Verbreitung des Unix-Codes profitieren. Stallman begann das Projekt GNU mit dem Schreiben von Komponenten des geplanten Systems, die ohne Änderung auch auf Unix-System laufen können sollten. Die Entwicklung von Werkzeugen für GNU konnte damit im Umfeld der Universitäten und anderer weltweiter Großrechenzentren voranschreiten.

Das Projekt war von enormer Größe. Irgendwie mußten freiwillige Programmierer gefunden und organisiert werden, die sich an die Arbeit machten, die Werkzeuge herzustellen, die für die spätere eigentliche Konstruktion benötigt wurden. Stallman selbst hat mehrere grundlegende Werkzeuge geschrieben. Andere wurden von kleinen oder großen Programmiererteams von anderswo beigetragen und mit Stallmans Projekt verbunden oder direkt verbreitet. Einige Orte im sich entwickelnden Netzwerk wurden zu Archiven dieser GNU-Komponenten, und in den 80er Jahren fanden die GNU-Werkzeuge Beachtung

und Akzeptanz bei Unix-Benutzern auf der ganzen Welt. Die Stabilität, Verlässlichkeit und Wartungsfreundlichkeit der GNU-Werkzeuge wurde sprichwörtlich, während Stallmanns überragendes Können als Designer den Entwicklungsprozeß gleichzeitig überholte und vorantrieb. Die Verleihung des MacArthur-Forschungspreises an Stallmann 1990 war die angemessene Anerkennung für seine konzeptionellen und technologischen Innovationen und deren soziale Konsequenzen.

Das Projekt GNU und die Stiftung Freie Software (Free Software Foundation), die es 1985 hervorbrachte, waren nicht die einzigen Quellen der Idee einer freien Software. Verschiedene Formen von Urheberrechtslizenzen, die freie oder teilweise freie Software fördern sollten, entstanden innerhalb der Universitäten, meistens im Umfeld von Unix. Die Universität von Berkeley begann mit dem Entwurf einer Unix-Version, die frei innerhalb der akademischen Gemeinschaft verbreitet werden sollte. Auch BSD Unix, wie es genannt wurde, behandelte AT&Ts Unix als den Standard. Der Programmtext war weithin bekannt und bildete ein Reservoir an Werkzeugen und Techniken, aber seine Lizenzbedingungen behinderten seine Ausdehnung auf weitere Anwendungen. Die Löschung hardware-spezifischer proprietärer Programminformation aus den Kopien bedeutete zudem, daß niemand mit BSD ein funktionierendes Betriebssystem für irgendeinen bestimmten Computer herstellen konnte. Andere Arbeiten an Universitäten wurden zu quasi freier Software; die graphische Benutzeroberfläche X-Windows für Unix-Systeme wurde am MIT entwickelt und wurde mit Source-Code und der Erlaubnis ausgeliefert, es frei zu verändern. Und 1989-1990 begann ein Informatik-Student an der Universität Helsinki das Projekt, das den Kreis schloß und der Vision von freier Software zum Durchbruch verhalf.

Was Torvalds tat war, daß er begann, ein Informatik-Lehrwerkzeug für das wahre Leben nutzbar zu machen. Andrew Tannenbaums MINIX - Kernel [19] war ein Stapel von Aufzeichnungen für einen Betriebssystem-Kurs, der ein Beispiel einfacher Lösungen für einfache Probleme war. Langsam und zunächst ohne die eigene Absicht wirklich zu erkennen, schrieb Linus den MINIX-Kernel zu einem wirklichen Kernel für Unix-Systeme um, die auf Intel x86-Prozessoren liefen, den Motoren, die weltweit Standard-PCs antreiben. Bei der Entwicklung des Kernels, den er Linux nannte, wurde ihm klar, daß sein Projekt die besten Chancen hätte, wenn er es so ausrichtete, daß die bestehenden GNU-Komponenten damit kompatibel sind.

Das Ergebnis von Torvalds Arbeit war 1991 die Veröffentlichung eines in den Grundzügen funktionierenden Modells eines freien Software-Kernels im Internet für ein Unix-ähnliches PC-Betriebssystem, der mit den leistungsfähigen von Stallmann geschaffenen und von der Free Software Foundation vertriebenen GNU-Komponenten voll kompatibel war und auf ihnen aufbaute. Weil Torvalds den Kernel unter der Allgemeinen Öffentlichen Lizenz (General Public Licence) der Free Software Foundation - dazu wird noch zurückzukommen sein - weitergab, konnten weltweit hunderte und Tausende von Programmierern sicher sein, daß ihre Beiträge zur Weiterentwicklung des Kernels zu einer für immer freien Software führen würden, die von niemandem als proprietäres Produkt vereinnahmt werden konnte. Jeder wußte, daß jeder andere die eigenen Beiträge testen, verbessern und die Verbesserungen verbreiten konnte. Tovalds hieß Beiträge willkommen und gab mit genialer Effizienz die Richtung vor, ohne Enthusiasmus zu bremsen. Die

Entwicklung des Linux-Kernels bewies, dass das Internet es möglich machte, weit größere Mengen an Programmierern zusammenzubringen, als es ein Hersteller vermag; diese waren fast ohne Hierarchie in einem Entwicklungsprojekt vereinigt, das letztlich mehr als eine Million Zeilen Computer-Code hervorbrachte - eine in dieser Dimension bisher unvorstellbare Art der Zusammenarbeit [20].

Bis 1994 hatte Linux die Version 1.0 erreicht, die einen verwendbaren Produktions-Kernel darstellte. Version 2.0 wurde in 1996 erreicht, und in 1998 - nunmehr mit Kernel-Version 2.2.0, die nicht nur für x86-Maschinen, sondern eine Vielzahl anderer Architekturen lieferbar war - waren die Kombination aus Linux-Kernel und einer umfangreicheren Sammlung von Komponenten aus dem Projekt GNU und Windows NT die einzigen zwei Betriebssysteme der Welt, die Marktanteile hinzugewannen. Eine interne Microsoft-Einschätzung der Situation, die im Oktober 1998 an die Öffentlichkeit gelangte und vom Unternehmen unverzüglich als echt anerkannt wurde, kam zu dem Schluß, daß "Linux ein qualitativ hochwertiges UNIX-System darstellt, dem für kritische Anwendungen vertraut wird und das wegen seines (!) offenen Source-Codes eine langfristige Glaubwürdigkeit erzielt, die viele andere konkurrierende Betriebssysteme nicht erreichen" [21]. Zu diesem Zeitpunkt wurden GNU/Linux-Systeme bereits überall auf der Welt in Anwendungen von Web-Servern großer E-Commerce-Web-Seiten bis zu "ad hoc-Supercomputer"-Bänken in der Netzwerk-Struktur großer Geldhäuser.

GNU/Linux wird auf dem Space Shuttle eingesetzt und läuft hinter den Kulissen auf Computern bei (ja !) Microsoft. Vergleichende Industriestudien zur Verlässlichkeit von UNIX-Systemen haben wiederholt gezeigt, daß Linux der bei weitem stabilste und zuverlässigste Unix-Kernel ist, dessen Verlässlichkeit nur von den GNU-Werkzeugen selbst übertroffen wird. GNU/Linux läßt nicht nur kommerzielle proprietäre Unix-Versionen für PCs hinter sich, sondern ist bekannt für seine Fähigkeit, ungestört und ohne Beschwerden, für Monate unter Vollastbedingungen ohne Absturz zu funktionieren.

Andere Bestandteile der Free-Software-Bewegung sind ebenso erfolgreich gewesen. Apache, das weltweit mit Abstand weitestverbreitete Web-Server-Programm, ist freie Software ebenso wie Pearl, die Programmiersprache, die von Web-Designern für anspruchsvolle Websites verwendet wird. Netscape Communications verbreitet nun seinen Netscape Communicator 5.0 als freie Software unter einer engverwandten Variante der General Public License der Free Software Foundation. Führende PC-Hersteller, darunter IBM haben Pläne angekündigt, GNU/Linux nach Kundenwunsch auf ihren als Web- und Fileserver eingesetzten Hochleistungsrechnern zu installieren, bzw. sind bereits dabei, dies zu tun. Samba, ein Programm, das GNU/Linux-Rechner dazu veranlaßt, wie Windows NT File Server zu funktionieren, ist weltweit als Alternative zu Windows NT Server im Einsatz und sorgt für effektiven Wettbewerb mit Microsoft auf dessen Heimatmarkt. Im Vergleich zu den Software-Standards, die in der Industrie bisher bekannt waren - und deren andauernde Relevanz Ihnen bei Ihrem nächsten Windows-Absturz bewußt werden wird - steht am Ende dieses Jahrhunderts eine unzweideutige Aussage: Die profitabelste und mächtigste Firma der Welt kommt als abgeschlagener Zweiter ins Ziel, der alle außer dem wahren Sieger aus dem Feld geschlagen hat. Die Beharrung auf eigenen Standards kombiniert mit kapitalistischer Gier hat jede kommerzielle Konkurrenz von Bedeutung zerstört, aber wenn es um gute Software geht, hat die Anarchie gewonnen.

### III. Anarchismus als Produktionsform

Es ist eine schöne Geschichte und wenn der IP-Druiden und der Wirtschaftszwerg nicht von der Theorie geblendet gewesen wären, hätten sie sie kommen sehen. Aber obwohl einige von uns dafür gekämpft haben und es jahrelang vorausgesagt haben, sind die theoretischen Konsequenzen so subversiv für die Denkstrukturen, in denen es sich die Zwerge und Druiden bequem gemacht haben, daß wir es ihnen ihre Ignoranz kaum zum Vorwurf machen können.

Die Fakten haben gezeigt, daß irgendetwas falsch war mit der Metapher der "Anreize", die das konventionelle Urheberrechts-Denken unterfüttert [22]. Aber die Fakten haben noch mehr gezeigt. Sie bieten einen Ausblick auf die Zukunft menschlicher Kreativität in einer Welt der globalen Verbundenheit, und das ist keine Welt für Zwerge und Druiden.

Bevor wir eine kurze Pause in der realen Welt eingelegt haben, wurde wie folgt argumentiert: Software - egal ob ausführbare Programme, Musik, visuelle Kunst, Liturgie, Waffen oder was auch immer - besteht aus Datenströmen, die - obwohl im wesentlichen nicht unterscheidbar - einer verwirrenden Vielzahl rechtlicher Kategorien unterworfen werden. Diese Vielzahl wird mit der Zeit instabil, was mit Gründen der Rechtsentwicklung als solcher zu tun hat. Die instabile Verschiedenheit der Regeln wird durch den Bedarf nach der Unterscheidung von Eigentumsinteressen an diesen Datenströmen verursacht. Diesen Bedarf haben vor allem die, die von den sozial akzeptierten Monopolen profitieren, welche sich dadurch gebildet haben, daß Ideen als Eigentum angesehen werden. Wer sich bei uns um die soziale Ungleichheit und kulturelle Hegemonie Sorgen macht, die von diesem intellektuell unbefriedigenden und moralisch verwerflichen Regime hervorgerufen werden, sieht sich gnadenlos ausgepiffen. Die, die pfeifen, die Zwerge und Druiden, glauben, daß diese Regeln des geistigen Eigentums nötig sind - aber nicht, weil jemand in Murdochland um Hilfe ruft - obwohl man jede Unterstützung gern entgegennimmt -, sondern weil die Metapher der "Anreize", die sie für ein wirkliches Argument halten, diese Regeln zur Produktion guter Software einfach erforderte, egal welche bedauernswerten Konsequenzen dies auch nach sich ziehe. Der einzige Weg, an diesem Glauben festzuhalten ist, die Fakten zu ignorieren. Inmitten der digitalen Revolution mit all ihren ausführbaren Datenströmen, die alles möglich machen, verbessern proprietäre Regime nicht nur nichts, sie können eine ganze Menge verschlimmern. Was immer auch sonst an Eigentumsregimen faul ist, in diesem Zusammenhang haben sie den Fortschritt nicht gefördert, sondern behindert.

Wie aber sieht diese mysteriöse Alternative aus? Freie Software existiert, aber was sind ihre Mechanismen und wie kann man diese mit Blick auf die digitale Gesellschaft verallgemeinern?

#### Die juristische Theorie freier Software

Es gibt einen Mythos, der wie alle Mythen teilweise auf Wahrheit beruht, nämlich daß alle Computerprogrammierer Libertäre sind. Rechtsgerichtete Programmierer sind

Kapitalisten, die an ihren Aktienoptionen hängen, aber Steuern, Gewerkschaften und Bürgerrechte verabscheuen; Linksgerichtete hassen den Markt und jede Form von Regierung, glauben an starke Verschlüsselung, egal zu wieviel nuklearem Terrorismus dies führen könnte, [23] und sie mögen Bill Gates nicht, weil er reich ist. Es mag Gründe geben, so zu denken. Aber der wichtigste Unterschied zwischen dem politischen Denken innerhalb und außerhalb der digitalen Welt ist, daß in der Netz-Gesellschaft Anarchismus (oder genauer besitz-ablehnender Individualismus) eine bedenkenswerte politische Philosophie darstellt.

Der Mittelpunkt des Erfolgs der Free-Software-Bewegung und die größte Leistung von Richard Stallman ist nicht ein Stück Computer-Code. Der Erfolg der freien Software, eingeschlossen der überwältigende Siegeszug von GNU/Linux, kommt aus der Fähigkeit, außerordentliche, hochwertige Leistungsbereitschaft für Projekte immenser Größe und Komplexität freizusetzen. Und diese Fähigkeit ist bedingt durch den juristischen Rahmen, innerhalb dessen diese Arbeitskraft mobilisiert wird. Als visionärer Designer hat Richard Stallman mehr als Emacs, GDB oder GNU geschaffen. Er schuf die General Public License.

Die GPL, [24] auch bekannt als "Copyleft", benutzt Urheberrecht, um - mit Toby Milsom gesprochen - die Phänomene des Anarchismus abzubilden. Die Präambel zur GPL drückt dies aus:

"Wenn wir über freie Software sprechen, beziehen wir uns auf Freiheit, nicht auf den Preis. Unsere General Public Licenses dienen dazu sicherzustellen, daß Du die Freiheit hast, Kopien freier Software zu verbreiten (und für diesen Service Geld zu verlangen, wenn Du willst), daß Du Source Code bekommst oder ihn bekommen kannst, wenn Du willst, daß Du die Software verändern kannst, oder Teile davon in neuen, freien Programmen verwenden kannst und daß Du weißt, daß Du diese Dinge tun darfst.

Um Deine Rechte zu sichern, müssen wir Restriktionen aufrichten, die jedermann verbieten, Dir diese Rechte abzusprechen oder Dich zu bitten, sie abzugeben. Diese Restriktionen führen zu gewissen Verantwortlichkeiten für Dich, wenn Du Kopien der Software verbreitest oder veränderst.

Wenn Du zum Beispiel Kopien eines solchen Programms verbreitest, ob gratis oder für eine Gebühr, mußt Du allen Empfängern die Rechte einräumen, die Du selbst hast. Du mußt sicherstellen, daß sie ebenfalls den Source Code bekommen oder bekommen können. Und Du mußt ihnen diese Bestimmungen zeigen, damit sie ihre Rechte kennen."

Viele Varianten dieser Grundidee freier Software sind in verschiedenen Lizenzmodellen ausgedrückt. Die GPL unterscheidet sich dennoch in einem wichtigen Punkt von anderen Formen, diese Werte auszudrücken. § 2 dieser Lizenz bestimmt in dem einschlägigen Passus:

"Du kannst Deine Kopie oder Kopien des Programms oder eines Teils davon verändern und dadurch ein Werk schaffen, das auf dem Programm basiert, und dieses Werk

kopieren und verbreiten..., vorausgesetzt, daß Du alle folgenden Bedingungen erfüllst:

...

b) Du mußt dafür sorgen, daß jedes von Dir verbreitete oder veröffentlichte Werk, das ganz oder zum Teil auf dem Programm basiert, für jedermann ohne Gebühren unter den Bedingungen dieser Lizenz lizenziert wird."

§ 2 b) der GPL wird oft als "einschränkend" bezeichnet, aber sein Ansatz ist freiheitsschaffend. Er schafft etwas Gemeinsames, zu dem jeder etwas beitragen, von dem aber niemand etwas wegnehmen darf. Wegen § 2b) kann jeder, der einen Beitrag zu einem GPL-Projekt leistet, sicher sein, daß er und alle Nutzer das Programm unbeschränkt nutzen, verändern oder weiterverbreiten können, daß der Source Code immer verfügbar sein wird und daß - anders als bei kommerzieller Software - seine Langlebigkeit nicht durch Marktzwänge oder durch Entscheidungen zukünftiger Programmierer beeinflußt wird. Diese "Vererblichkeit" der GPL ist manchmal als Ausdruck der anti-kommerziellen Voreingenommenheit der Free-Software-Bewegung kritisiert worden. Nichts könnte unwahrer sein. Der Effekt von § 2 b) ist es, kommerzielle Verbreiter freier Software zu besseren Wettbewerbern proprietärer Software-Unternehmen zu machen. Zur Bestätigung dieser Behauptung kann man nichts Besseres tun, als die proprietären Wettbewerber zu fragen. Der Autor von Microsofts "Halloween"-Memorandum, Vinod Valloppillil, hat es so ausgedrückt:

"Die GPL und ihre Abneigung gegen Code-Piraterie versichert Kunden, daß sie nicht in eine Entwicklungs-Sackgasse geraten, indem sie eine kommerzielle Version von Linux verwenden.

Die Entwicklungs-Sackgasse ist der Kern des Software-FUD Arguments [\[25\]](#)."

Aus dem Microspeak übersetzt bedeutet dies, daß die Strategie, mit der der marktbeherrschende proprietäre Softwarehersteller seine Kunden von Wettbewerbern fernhält - nämlich durch das Säen von Angst und Zweifel (Fear, Uncertainty, Doubt (FUD)) hinsichtlich der langfristigen Überlebenschancen der Software - im Bezug auf GPL-Programme ineffektiv ist. Nutzer von GPL-Code, einschließlich derjenigen, die diese Software von kommerziellen Weiterverbreitern beziehen, wissen, daß zukünftige Verbesserungen und Reparaturen für alle aus dem gemeinsamen Ganzen angeboten werden und brauchen weder das Verschwinden ihres Lieferanten noch das Szenario zu fürchten, in dem ein Einzelner eine besonders attraktive Verbesserung oder eine wichtige Fehlerkorrektur gegen Geld vermarktet.

Urheberrechtsregeln zu verwenden, um ein gemeinsames Ganzes im Cyberspace zu schaffen, ist die zentrale institutionelle Struktur, die den anarchistischen Triumph ermöglicht. Durch die Sicherstellung von freiem Zugang und Veränderungsmöglichkeiten auf jeder Stufe des Prozesses bedeutet, daß die Entwicklung von Software sich auf die schnelle Lamarcksche Art vollzieht: jedes erfolgreich aufgenommene Stück fremder Arbeit kann direkt übernommen werden. Dadurch konnte etwa der Linux-Kernel alle seiner

proprietären Vorgänger überholen. Weil Umgehungen unmöglich sind, sind Trittbrettfahrer willkommen. Dies löst eines der zentralen Rätsel kollektiver Aktivität in einem proprietären Sozialsystem.

Nicht-proprietäre Produktion ist ebenso direkt verantwortlich für die berühmte Stabilität und Verlässlichkeit freier Software, die aus dem von Eric Raymond so genannten "Linus-Gesetz" entsteht: Bei einer ausreichend hohen Zahl wachsamer Augen sind alle Fehler trivial. Praktisch bedeutet freier Zugang zum Source Code: Wenn ich ein Problem habe, repariere ich es. Weil ich es reparieren kann, brauche ich es so gut wie nie zu tun, weil es fast immer schon jemand gesehen und repariert hat.

Für die Free-Software-Gemeinschaft ist anarchistische Produktion nicht nur eine moralische Verpflichtung. Wie Richard Stallman schrieb: Es geht um Freiheit, nicht um den Preis. Zudem ist es eine Frage der Nützlichkeit, das Streben danach, bessere Software zu produzieren als proprietäre Modelle sie hervorbringen könnten. Vom Standpunkt des Druiden aus mag Copyleft die Perversion der Theorie sein, aber es löst besser als jeder andere Vorschlag der letzten Jahrzehnte das Problem der Anwendung von Urheberrecht auf Computerprogramme, bei denen funktionelle und expressive Eigenschaften so sehr miteinander verwoben sind. Daß es außerdem bessere Software hervorbringt als die Alternative muß kein Grund sein, von nun an traditionelle Urheberrechtsprinzipien auch für diejenigen zu verbieten, die weiterhin minderwertige Software besitzen und vermarkten wollen oder (gnädiger) für diejenigen, deren Produkte einen zu kleinen Anwendungsbereich haben, um kollektiv produziert zu werden. Aber unsere Geschichte soll den Druiden als Warnung dienen: Die Welt der Zukunft wird wenig mit der der Vergangenheit gemein haben. Die Regeln werden heute in zwei Richtungen umgebogen. Die Unternehmen, die "kulturelle Ikonen" oder anderes besitzen, und nun für ihre Autoren noch längere Schutzzeiten durchzusetzen versuchen, haben natürlich Wasser auf die druidischen Mühlen gelenkt [26]. Wer hat den Druiden denn die Konzertkarten bezahlt? Aber während die proprietäre Position versucht, sich immer weiter in einem Copyright-Konzept zu etablieren, das im Begriff ist, von kleineren Unannehmlichkeiten wie begrenzten Schutzzeiten und "Fair Use" (unbeschränkbare Verwertungsrechte) befreit zu werden, hat ganz im Mittelpunkt unseres Systems "kultureller Software" der anarchistische Gegenschlag schon begonnen. Und es soll für die Druiden noch schlimmer kommen, wie wir sehen werden. Aber zuvor müssen wir noch mit den Zwergen abrechnen.

### **Weil es da ist: Faradays Magnet und die menschliche Kreativität**

Letztlich verdienen die Zwerge noch eine Antwort. Warum machen Menschen freie Software, wenn sie daraus keinen Profit ziehen können? Normalerweise wurden darauf zwei Antworten gegeben. Eine stimmt zur Hälfte, eine ist falsch, aber beide sind unbefriedigend einfach.

Die falsche Antwort ist eingebettet in verschiedene Bezugnahmen auf die "Hackerkultur des Gebens und Nehmens". Diese Benutzung ethnografischen Jargons hat vor einigen Jahren Einzug in die Auseinandersetzung gehalten und setzte sich schnell, wenngleich irreführend, durch. Sie zeigt nur, daß die Wirtschaftsfetischisten unseren Denkprozess so sehr korrumpiert haben, daß wir bereit sind, jede Form nicht marktgemäßen Verhaltens

mit jeder anderen gleichzusetzen. Dabei ist Geben und Nehmen, wie der marktmäßige Tauschprozess, eine Institution der proprietären Welt. Reziprozität spielt eine zentrale Rolle bei diesen symbolischen Aufführungen natürlicher Abhängigkeit, und wenn der Fisch zu wenig wiegt, gibt es Ärger. Freie Software steht, wie schon wiederholt gesagt wurde, jedem zur Verfügung. Hier spielt sich kein Ritual der Reziprozität ab. Einige Menschen geben Programmtext frei, den andere verkaufen, benutzen, verändern oder im Ganzen ausleihen, um Stücke daraus für andere Zwecke zu verwenden. Ohne die sehr große Zahl von Leuten (mehrere zehntausend höchstens) geringzuschätzen, die zu GNU/Linux beigetragen haben, ist diese doch um einige Potenzen geringer als die derjenigen Nutzer, die keinerlei Beitrag geleistet haben [27].

Ein Teil der richtigen Antwort liegt in der Behauptung, daß freie Software von Menschen gemacht wird, die ihren Ruf ausbauen möchten. Berühmte Linux-Hacker - so die Theorie - sind auf der ganzen Welt als Programmiergenies bekannt. Daraus ziehen sie entweder ein erhöhtes Selbstbewußtsein oder indirekten wirtschaftlichen Vorteil [28]. Aber die Programmier-Genies, so sehr sie auch zur freien Software beigetragen haben mögen, haben nicht die Hauptarbeit geleistet. Eine Reputation entsteht, wie Linus Torvalds selbst oft gesagt hat, aus dem Anerkenntnis, daß die ganze Arbeit von anderen erledigt wurde. Außerdem hat die Free-Software-Bewegung, wie viele Beobachter erkannt haben, auch Dokumentationen auf höchstem Niveau hervorgebracht. Das Schreiben von Dokumentation gehört nicht zu dem, was Hacker cool finden, und meistens waren die Autoren von Dokumentationen andere als die, die Code geschrieben haben. Andererseits darf man die indirekten materiellen Vorteile von größerer Reputation nicht unterschätzen. Die meisten Autoren freier Software, die ich kenne, arbeiten tagsüber in der Technologieindustrie und die Fertigkeiten, die sie bei ihrer kreativen Tätigkeit außerhalb des Marktes zeigen, erhöhen nicht selten meßbar ihren Wert in der Arbeitswelt. In dem Maße, in dem die Free-Software-Bewegung die kritische Masse erreicht und sich als Basis für eine ganze Reihe neuer Geschäftsmodelle neben der kommerzielle Verbreitung von Dingen, die man auch umsonst bekommen kann, etabliert, wird eine immer größere Zahl von Menschen angestellt, um freie Software zu programmieren. Aber um in diesem Feld Anstellung zu bekommen, müssen sie darin bereits eingeführt sein. Daraus entsteht natürlich eine Motivation, freie Software zu machen, aber das ist nicht die ganze Erklärung.

In der Tat ist der Rest der Antwort zu simpel, um die nötige Beachtung zu finden. Am besten versteht man dies, indem man der kurzen und wenig beachteten Karriere eines anfangs widerwilligen Free-Software-Programmierers folgt. Der Microsoft-Mitarbeiter Vinod Valloppillil, kaufte und installierte ein Linux-System auf einem seiner Büro-Computer, um eine vergleichende Analyse von Linux zu verfassen. Diese ist als zweites der berühmten "Halloween"-Memoranden bekannt geworden. Valloppillil hatte Schwierigkeiten, weil die (kommerzielle) Linux-Version, die er verwendete, keinen Dämon beinhaltete, um mit dem DHCP-Protokoll zur Vergabe dynamischer IP-Adressen umzugehen. Das Ergebnis ist so wichtig, daß wir uns ein weiteres Mal dem Schreibstil von Microsoft aussetzen sollten:

"Eine kleine Anzahl von Webseiten und FAQs später fand ich eine FTP-Seite mit einem Linux-DHCP-Treiber. Der Treiber war von einem bei Fore Systems angestellten Ingenieur geschrieben worden (dies ergab sich aus seiner e-mail-Adresse, obwohl ich glaube, daß er den Treiber in seiner Freizeit geschrieben hat).

Eine weitere Sammlung von Dokumentationen und Gebrauchsanweisungen war von einem *ungarischen* Hacker geschrieben worden und enthielt relativ einfache Anweisungen wie der Treiber zu installieren war.

Ich lud den Treiber herunter, dekomprimierte ihn und schrieb zwei einfache Kommandos:

Make - kompiliert die Binary-Files des Treibers

Make Install - installiert die Files als ein Linux Dämon

Das Eingeben von "DHCPD" (für DHCP Client Daemon) in die Kommandozeile startete den Prozess und - siehe da - schon funktionierte das IP-Netzwerk.

Weil ich den Treiber-Code gerade heruntergeladen hatte, spielte ich damit ein bißchen herum. Obwohl er nicht so ausbaufähig war wie der DHCP-Treiber, den wir mit NT5 ausliefern (er fragt etwa nicht nach Zufalloptionen und kann Ergebnisse nicht speichern), war mir sofort klar, wie ich das Programm ergänzen konnte, um diese Zusatzfunktionen einzubauen. Der ganze Treiber bestand aus etwa 2.600 Zeilen.

Ein Beispiel von esoterischer Funktionserweiterung war eine offenbar von einem Dritten eingebaute Sammlung von Routinen, die die DHCP-Abfrage mit host-spezifischen Zeichenfolgen auspolstern, die für Kabel-Modem- und ADSL-Seiten nötig sind.

Einige weitere Schritte waren nötig, um den Treiber dazu zu bringen, automatisch zu starten und meine Ethernet-Karte zu konfigurieren, aber diese waren in dem Programmtext und in der Dokumentation des ungarischen Entwicklers beschrieben.

Ich bin ein armseliger UNIX-Programmierer, aber es wurde mir sofort klar, wie ich den DHCP-Treibercode entscheidend erweitern konnte (das Gefühl war umwefend und machte süchtig).

Außerdem, dank GPL und einer vollen Entwicklungsausrüstung an meinem Arbeitsplatz, hätte ich sofort meine Veränderungen aufschreiben und sie per e-mail innerhalb einiger Stunden verbreiten können (im Gegensatz zum Ablauf solcher Dinge bei NT). Dies zu tun hätte mich für ein größeres, ehrgeizigeres Linux-Projekt in der Zukunft vorbereitet. [29]

"Das Gefühl war umwefend und machte süchtig". Maschinen stop ! Microsoft bestätigt Mogens Metaphorische Ableitung von Faradays Gesetz. Wickle das Internet um jedes Gehirn des Planeten und versetze ihn in Drehung. Software fließt in den Drähten. Es ist eine Ur-Eigenschaft des menschlichen Geistes kreativ zu sein. "Dank GPL", wie Vallopillil richtig ausführte, ermöglichte ihm freie Software eine umwerfende Zunahme seiner eigenen Kreativität solcher Art, wie er sie in seinem täglichen Job bei dem "Größten Programmierunternehmen der Welt" nicht erreichen konnte. Wenn er nur diese erste Verbesserung losgeschickt hätte - wer weiß wo er heute wäre ?

Somit ist es, meine zwerghaften Freunde, am Ende doch nur ein menschliches Ding. Warum singt Figaro, warum schreibt Mozart Musik, die Figaro singt, warum erfinden wir alle neue Worte? Weil wir es können. Homo ludens, trifft Homo faber. Die soziale Bedingung globaler Zusammenschaltung, die wir das Internet nennen, ermöglicht uns allen kreativ zu sein auf eine neue und nie erträumte Weise. Wenn wir nur nicht "Eigentum" dazwischenfunken lassen. Wiederholt es mit mir, Zwerge und Menschen: Widerstehe dem Widerstand!

#### IV. Ihre Lordschaften sterben im Dunkeln?

Für den IP-Druiden, frisch aus dem Flugzeug von einer Woche in Bellagio, bezahlt von der Dreamworks AG, führt dies bereits zu Verdauungsproblemen.

Die Möglichkeiten menschlicher Kreativität dadurch freizusetzen, daß man jeden mit jedem verbindet? Das Eigentums-System beiseite schieben, damit wir alle unsere Stimmen dem Chor hinzufügen können - selbst wenn das bedeutet, daß wir unseren Gesang mit dem "Mormonischen Tabernakel" unterlegen und das Ergebnis einem Freund schicken? Keiner mehr, der zähnefletschend vor einer TV-ausgestrahlten Mischung aus Gewalt und ständiger Kopulation sitzt, die so schön zusammengestellt wurde, um das Interesse des jungen Mannes für die folgende Bierwerbung zu erhöhen? Was wird aus der Zivilisation? Was wird zumindest aus den Urheberrechts-Professoren?

Aber vielleicht ist dies voreilig. Bisher habe ich nur über Software gesprochen. Über richtige Software, die alte, die Computer antreibt. Nicht über diejenige, mit der DVD-Player funktionieren oder die, die von Grateful Death gemacht wird.

"Ach ja, Grateful Death, da war doch etwas Komisches mit denen, oder? Sie haben das Mitschneiden ihrer Konzerte nicht verboten, es machte ihnen nichts aus, wenn die Fans der Plattenindustrie ein Schnäppchen schlugen. Ihnen scheint es trotzdem gut zu gehen, muß man zugeben. Senator Patrick Leahy, ist der nicht ein früherer Death-Fan? Ich würde mich wundern, wenn der dafür stimmt, die Schutzzeiten von rechteevertenden Unternehmen auf 125 Jahre zu verlängern, damit Disney die Maus nicht in 2004 verliert. Und diese DVD-Player - das sind doch Computer, oder?"

In der digitalen Welt ist alles vernetzt. Langfristig kann es nicht darauf ankommen, einen Datenstrom vom anderen unterscheiden zu müssen, um das anwendbare Recht herauszufinden. Was der Software passierte, passiert heute bereits der Musik. Die Lordschaften der Musikindustrie strampeln heute schon wie wild, um die Kontrolle über die Verbreitung zu behalten, wenn sowohl Musikern als auch Hörern klar wird, daß die Vermittlungsinstitution nicht mehr nötig ist. Das Große Potemkinsche Dorf von 1999, die sogenannte "Initiative Sichere Digitale Musik" (Secure Digital Music Initiative) wird in sich zusammenfallen, lange bevor der erste Internet-Präsident in sein Amt eingeführt wird - und das aus ganz einfachen technischen Gründen, die denen, die Bescheid wissen genau so klar sind wie denen, die den Triumph freier Software diktiert haben [30]. Die anarchistische Revolution in der Musik unterscheidet sich von der bei der Software, aber

auch hier setzten sich die Fakten gegen die Theorie durch, wie jeder Teenager mit einer MP3-Kollektion eigenverbreiteter Musik unbekannter Künstler bestätigen wird. Ob man Mick Jagger ist oder ein großer nationaler Artist aus der Dritten Welt, der nach einem weltweiten Publikum sucht, oder ein Hinterhof-Künstler, der die Musik neu erfindet - bald hat die Musikindustrie ihnen nichts mehr anzubieten, was sie nicht besser umsonst bekommen können. Musik klingt nicht schlechter, wenn sie umsonst angeboten wird, bezahlt was Ihr wollt direkt an den Künstler und zahlt nichts, wenn Ihr nicht wollt. Gebt es Euren Freunden, vielleicht mögen die es.

Was bei der Musik geschieht, steht auch den Nachrichten bevor. Die Nachrichtenagenturen haben, wie jeder amerikanische Jura-Student sogar vor dem fast-obligatorischen Urheberrecht-für-Druiden-Kurs lernt, ein schützenswertes Eigentumsinteresse an ihren Nachrichtenlieferungen, wenn auch nicht an den Fakten, die sie berichten [31]. Warum also geben sie jetzt alle ihre Meldungen frei ? Weil Nachrichten in der Welt des Netzes Jedermanns-Nachrichten sind. Der ursprüngliche Vorteil der Nachrichtenleute, besser als andere untereinander verbunden zu sein als Kommunikation teuer war, besteht nicht mehr. Was heute zählt ist, Augäpfel zu sammeln und sie den Werbetreibenden zu liefern. Es sind nicht die Nachrichtenagenturen, die bei der Berichterstattung aus Kosovo Vorteile hatten. Noch weniger die Paragonen des "geistigen" Eigentums, Ihre Fernseh-Lordschaften. Jene mit ihren überbezahlten gutaussehenden Korrespondenten und ihrer unhandlichen technischen Infrastruktur sind so gut wie die einzigen Organisationen der Welt, die es sich nicht leisten können, immer überall präsent zu sein. Und dann müssen sie sich auch noch auf neunzig Sekunden pro Geschichte beschränken, weil sonst die Augäpfel-Jäger woanders hingehen. Also - wer macht bessere Nachrichten, die Proprietären oder die Anarchisten ? Wir werden es bald wissen.

Oscar Wilde sagte irgendwo, daß das Problem des Sozialismus ist, daß er zuviele Abende in Anspruch nimmt. Die Probleme mit dem Anarchismus als sozialem System liegen ebenfalls in den Transaktionskosten. Aber die digitale Revolution verändert zwei Aspekte politischer Ökonomie, die bis dahin über die gesamte menschliche Geschichte unverändert geblieben sind. Jede Software hat in der Welt des Netzes keinerlei Grenzkosten, während die Kosten sozialer Koordination sich soweit reduziert haben, daß die rasche Formierung und Auflösung großer und hochdiversifizierter sozialer Gruppierungen ohne geographische Einschränkung möglich ist [32]. Solche umwälzenden Veränderungen in den materiellen Lebensumständen haben notwendig auch ebensolche Veränderungen in der Kultur zur Folge. Zweifel ? Fragen Sie die Hirokesen ! Und natürlich bedrohen solche kulturellen Veränderungen die existierenden Machtbeziehungen. Zweifel ? Fragen Sie die Chinesische Kommunistische Partei. Oder warten Sie noch 25 Jahre und schauen Sie, ob Sie sie noch finden können, um die Untersuchung durchzuführen.

In diesem Zusammenhang ist es weder unvorhersehbar noch tragisch, daß die Zeiten des IP-Druiden vorbei sind. Tatsächlich kann dieser sich in die Wüste versetzt fühlen, einem imaginären Raum seine profitabel komplizierten Regeln erklärend, die eine nicht mehr bestehende Welt betreffen. Aber wenigstens ist er in angenehmer Gesellschaft, man kennt sich von all den glitzernden Parties in Davos, Hollywood und Brüssel. Unsere Medien-Lords spüren den Hauch ihres Schicksals, wie sehr sie auch glauben mögen, daß die Kraft mit ihnen sei. Die Regeln über Datenströme sind nurmehr von zweifelhafter Nützlichkeit,

Macht durch die Einbindung menschlicher Kreativität zu sichern. Im Lichte der Fakten klar gesehen, haben diese Kaiser noch weniger Kleider an, als die Models, die sie benutzen, um unsere Augäpfel zu fesseln. Obgleich gestützt von benutzer-unfreundlicher Technologie, einer Kultur vollständiger Überwachung, die es erlaubt, jeden Leser von "Eigentum" zu speichern und zur Kasse zu bitten und Rauchschwaden von Druiden-Atem, die jedem Jugendlichen vermitteln, jede Kreativität verschwände ohne die wohlwollende Herrlichkeit von BillG dem Schöpfer, Lord Murdoch von Überall und dem Lord Heigh Mouse, ist ihr Regnum bald zuende. Aber was auf dem Spiel steht, ist die Kontrolle über die knappste Ressource überhaupt: unsere Aufmerksamkeit. Diese zu binden verspricht in der digitalen Welt jeden Reichtum und die Lords werden darum kämpfen. Gegen sie sind nur die Anarchisten aufgestellt: Niemande, Hippies, Amateure, Liebhaber und Künstler. Der entstehende ungleiche Kampf ist die große politische und ökonomische Frage unserer Zeit. Aristokratie sieht schwer zu schlagen aus, aber das war ebenso in in 1788 und 1913. Es ist, wie Chou En-Lai über die französische Revolution sagte, zu früh, eine Prognose abzugeben.

### **Über den den Autor:**

Eben Moglen ist Professor für Recht und Rechtsgeschichte an der Columbia Law School in New York, USA. E-mail: [moglen@columbia.edu](mailto:moglen@columbia.edu). Der Text wurde übersetzt von [Andreas von Bonin](#), LL.M.

### **Danksagung**

Dieser Aufsatz liegt meinem Beitrag auf der Buchanan International Conference über Recht, Technologie und Information, Universität von Tel Aviv im Mai 1999, zugrunde; mein Dank gilt den Organisatoren für ihre freundliche Einladung. Ich schulde Pamela Karlan so viel wie immer für ihr Verständnis und ihre Ermutigung. Ich möchte insbesondere allen Programmieren weltweit danken, die freie Software möglich gemacht haben.

### **Fußnoten:**

1. Im ursprünglichen Zusammenhang handelte es sich um eine nur ungefähre Unterscheidung. In den späten 60er Jahren wurden verschiedene Grundfunktionen der Hardware durch Programme kontrolliert, die direkt in der Elektronik von Computerbauteilen enthalten waren und nicht mehr verändert werden konnten, nachdem das Gerät die Fabrik verlassen hatte. Solche symbolischen aber unveränderlichen Komponenten waren als "Microcode" bekannt, wurden dann aber überwiegend "firmware" genannt. Das "Soft-" in Software bezog sich - wie der Begriff "firmware" zeigt - zunächst auf die Fähigkeit des Nutzers, die Anweisungen zu verändern, die das Maschinenverhalten bestimmen. Wenn nun die digitale Revolution weitgehend dazu geführt hat, daß Computer von technisch inkompetenten Nutzern bedient werden, wird der Großteil traditioneller Software - Anwendungen, Betriebssysteme, numerische Kontrollinstruktionen usw.- für die meisten zur "firmware". Sie mag zwar eher symbolisch als elektronisch konstruiert sein, aber die Nutzer könnten sie nicht verändern, selbst wenn sie es - wie einige hilflos und bedauernd feststellen - wollten. Diese "Erhärtung" von Software ist eine Vorbedingung für

den proprietären Ansatz der rechtlichen Organisation digitaler Gesellschaft, von dem dieser Beitrag handelt.

2. Innerhalb der heutigen Generation verändert sich der Begriff sozialer "Entwicklung" von dem Besitz schwerer Industrie, die auf dem Verbrennungsmotor basiert, zu "post-Industrie", die auf digitaler Kommunikation und den damit verbundenen wissenschaftsgestützten Wirtschaftsformen aufgebaut ist.

3. Tatsächlich sind, wie ein Moment der Überlegung uns nahelegt, unsere Gene "firmware". Die Evolution vollzog den Übergang von analog zu digital schon bevor die fossile Zeitrechnung beginnt. Aber wir hatten keine Möglichkeit, dieses Programm gezielt zu modifizieren. Bis vorgestern. Im nächsten Jahrhundert werden auch Gene Software werden und wengleich ich dies hier nicht weiter thematisiere, sind die Konsequenzen unfreier Software in diesem Zusammenhang noch viel besorgniserregender als im Zusammenhang mit künstlerischen Leistungen.

4. Siehe z.B. *J. M. Balkin, 1998. Cultural Software: a Theory of Ideology.* New Haven: Yale University Press.

5. Vgl. *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. Normalerweise gefällt mir das Einfügen autobiografischer Elemente in die Lehre nicht. Aber weil es hier meine traurige Pflicht und meine große Freude ist, die Qualifikation bzw. den guten Glauben von mehr oder weniger jedermann herauszufordern, muß ich die Einschätzung meiner eigenen Qualifikation ermöglichen. Ich kam mit dem Handwerk der Computerprogrammierung zuerst in 1971 in Berührung. Ich verdiente als kommerzieller Programmierer im Alter von 13 Jahren mein erstes Gehalt und bestritt weiterhin in einer Vielzahl von Computerdiensten, Engineering und internationalen Technologiefirmen meinen Lebensunterhalt bis 1985. 1975 half ich beim Schreiben eines der ersten netzwerkgestützten e-mail-Systeme der USA mit; von 1979 an war ich beschäftigt mit der Forschung und Entwicklung fortgeschrittener Programmiersprachen bei IBM. Diese Tätigkeiten ermöglichten mir wirtschaftlich das Geschichts- und Jurastudium. Mein Einkommen ermöglichte mir, meine Studiengebühren zu bezahlen, aber nicht - um einem früher oder später zu erwartenden Argument der Wirtschaftszweige vorzugreifen - weil meine Programme geistiges Eigentum meines Arbeitgebers gewesen wären, sondern eher weil sie die Hardware, die mein Arbeitgeber verkaufte, besser machten. Das meiste was ich schrieb war grundsätzlich freie Software, wie wir sehen werden. Obwohl ich immer wieder einige wenig bedeutsame technische Beiträge zur Free-Software-Bewegung machte, die in diesem Beitrag beschrieben wird, waren meine Haupttätigkeiten dafür juristischer Natur: Ich bin seit fünf Jahren (natürlich unbezahlt) der Rechtsberater der Stiftung Freie Software (Free Software Foundation).

7. Der CD-Spieler hat natürlich weitere Zu- und Ausgänge in Kontrollkanälen: Schalter und Infrarot-Signale sind Input, und Zeit- und Titelanzeige sind Output.

8. Dies ist keine Einsicht, die nur für den hier behandelten Gegenstand zutrifft. Eine eng

verwandte Idee bildet eines der wichtigsten Prinzipien in der Geschichte des angelsächsischen Rechts, unerreicht ausgedrückt von Toby Milsom im folgenden:

"Das Leben des Common Law spielte sich im Mißbrauch seiner Grundideen ab. Wenn die Eigentumsregeln eine Antwort gaben, die heute ungerecht erscheint, wurde auf Schuldrecht ausgewichen; und die Billigkeitsrechtsprechung hat bewiesen, daß mit den Instrumenten des Schuldrechts die Phänomene des Sachenrechts bekämpft werden können. Wenn die Vertragsregeln eine ungerechte Antwort gaben, wurde auf das Deliktsrecht ausgewichen... Wenn die Regeln eines Deliktstatbestands, etwa Täuschung, eine ungerechte Antwort gaben, konnte man einen anderen, etwa Fahrlässigkeit probieren. Und so drehte sich die juristische Welt."

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

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

10. Vgl. [The Virtual Scholar and Network Liberation](#).

11. Einiges Grundvokabular soll hier erklärt werden: Digitale Computer führen numerische Instruktionen aus, d.h. Bit-Ströme, die Informationen in der "Mutter-"Sprache enthalten, die der Entwickler der Maschine geschaffen hat. Diese wird normalerweise als "Maschinensprache" bezeichnet. Die Maschinensprachen sind auf die Ausführungsgeschwindigkeit auf der Hardwareebene ausgerichtet und eignen sich nicht für den direkten Gebrauch durch Menschen. Also gehören zu den zentralen Computerkomponenten die "Programmiersprachen", welche Ausdrücke, die für Menschen Sinn machen, in Maschinensprache übersetzen. Die üblichste und wichtigste, aber bei weitem nicht die einzige Form von Computersprachen ist der "Compiler". Der Compiler führt statische Übersetzungsarbeit aus, so daß eine Datei, die Source-Code, also menschenverständliche Anweisungen, enthält, in eine oder mehrer Dateien mit Maschinensprache übersetzt wird, die als "object code" bezeichnet wird.

12. Dies war, wie ich sagen sollte, das Hauptfeld meiner Forschung im Zusammenhang mit einer APL (A Programming Language) genannten Programmiersprache und ihren Nachfolgern. Aus Gründen, die später erläutert werden, hat sich dieser Ansatz nicht durchgesetzt.

13. Diese Beschreibung verlangt nach einigen Details. In der Mitte der 70er Jahre sah sich IBM beachtlicher Konkurrenz bei Großrechnern ausgesetzt, während aufgrund einer Wettbewerbsentscheidung der US-Wettbewerbsgerichte IBM verpflichtet wurde, Software "ungebündelt", also auf separate Rechnung zu liefern. In diesem weniger wichtigen Sinn hörte daher Software auf, frei zu sein. Aber - ohne auf die seinerzeit heiße, aber heute abgeschlossene Diskussion über IBMs Softwarepreis-Politik zurückkehren zu wollen - die Entbündelungsverpflichtung hatte weniger Auswirkungen auf die sozialen Praktiken bei der Softwareproduktion, als angenommen werden könnte. So konnte ich als Mitverantwortlicher für die technische Verbesserung eines

Programmiersprachen-Produkts bei IBM von 1979-1985, dieses als "fast freie" Software behandeln. Ich konnte mit Nutzern deren Verbesserungen diskutieren und mit ihnen gemeinsam Entwicklungen durchführen zum Nutzen aller anderen Benutzer.

14. Diese Beschreibung ist hoch komprimiert und wird für diejenigen, die damals ebenfalls in der Branche arbeiteten übersimplifiziert und zu rosig erscheinen. Urheberrechtsschutz für Computersoftware war ein kontroverses Thema in den 70er Jahren, das zu der berühmten CONTU-Kommission und ihren vorsichtig für den Urheberrechtsschutz eintretenden Empfehlungen von 1979 geführt hat. Und IBM schien weit weniger kooperativ mit seinen Nutzern umzugehen, als es hier dargestellt wird. Aber das wichtigste Element ist hier der Kontrast mit der Welt des PC, des Internet, der Dominanz von Microsoft und dem daraus entstehenden Impetus für die Free-Software-Bewegung. Ich konzentriere mich daher hier auf die Elemente, die diesen Kontrast ausdrücken.

15. Ich diskutiere den Einfluß von PC-Software in diesem Zusammenhang, die Entwicklung des "Marktes um Augäpfel" und des "gesponsorten Lebens" in anderen Kapiteln meines im Erscheinen begriffenen Buches *Das Unsichtbare Barbecue*, aus dem auch dieser Artikel ein Teil ist.

16. Dasselbe Muster von Widersprüchlichkeit, bei dem zu weitreichender Instabilität führende schlechte Programmierleistungen gleichzeitig beängstigend und bestätigend auf technisch Inkompetente wirken, zeigt sich in der vornehmlich amerikanischen Erscheinung der Jahr-2000-Hysterie.

17. Die kritischen Implikation dieser einfachen Beobachtung werden ausgeführt in "Wie man nicht über 'das Internet' denken sollte", in *Das Unsichtbare Barbecue*, im Erscheinen.

18. Technisch Versierte werden wiederum erkennen, daß hier die Geschehnisse zwischen 1969 und 1973 komprimiert geschildert werden.

19. Betriebssysteme, sogar Windows (das dies vor seinen Nutzern so gut wie möglich zu verstecken sucht) sind tatsächlich eher Sammlungen einzelner Komponenten als ungeteilte Einheiten. Das meiste von dem, was ein Betriebssystem tut (verwalten von Dateisystemen, Prozessausführungen kontrollieren usw.) kann von den Spezifikationen der Computerhardware getrennt werden, auf der das Betriebssystem läuft. Nur ein kleiner innerer Teil muß sich tatsächlich mit den exzentrischen Besonderheiten einer bestimmten Hardware auseinandersetzen. Ist das Betriebssystem einmal in einer bestimmten Sprache, etwa in C, geschrieben, muß nur der innerste Teil, der Kernel, an eine bestimmte Hardware-Architektur angepaßt werden.

20. Eine vorsichtige und kreative Analyse über die Art, wie Torvalds diesen Prozeß möglich machte und was für Implikationen er für die soziale Praxis der Softwareentwicklung hat, findet sich bei *Eric S. Raymond* in einer Seminararbeit von 1997, [The Cathedral and the Bazaar](#), die selbst eine erhebliche Rolle bei der Ausbreitung der Idee freier Software gespielt hat.

21. Dies ist ein Zitat aus dem als "Halloween Memo" bekannten Papier, das bei Eric

Raymond, dem es zugespielt wurde, gefunden werden kann,

<http://www.opensource.org/halloween/halloween1.html>.

22. Nicht früher als 1994 informierte mich ein talentierter und technisch kompetenter (obgleich Windows benutzender) Rechts- und Wirtschaftsstudent einer großen US-amerikanischen Rechtsfakultät vertraulich darüber, daß freie Software nicht existieren könne, weil niemand den Anreiz habe, wirklich ausgefeilte Programme, die viel Investition erfordern, herzustellen, nur, um sie danach zu verschenken.

23. Diese Frage verdient ebenfalls eine besondere Prüfung, weil sie so verkrustet ist mit Bemerkungen zur Macht der Einzelstaaten. Dazu meinen kurzen Essay, "[So Much for Savages: Navajo 1, Government 0 in Final Moments of Play.](#)"

24. Vgl. [GNU General Public License, Version 2, June 1991.](#)

25. [V. Valloppillil, Open Source Software: A \(New?\) Development Methodology.](#)

26. Das drohende Auslaufen des Disney-Eigentums an der Maus verlangt, aus der Sicht dieses reichen "Wahlkampfspenders", zum Beispiel die Änderung des allgemeinen Urheberrechts der Vereinigten Staaten. Siehe dazu "Darf es ein bisschen mehr sein? Das Verpulvern öffentlicher Güter" in *Das Unsichtbare Barbecue*, im Erscheinen.

27. Eine aktuelle Industrieschätzung beziffert die weltweit eingesetzten Linux-Systeme mit 7.5 Millionen. Siehe *Josh McHugh*, 1998. "[Linux: The Making of a Global Hack.](#)" *Forbes* (August 10). Weil die Software aber im Netz frei verfügbar ist, sind Schätzungen über den wirklichen Gebrauch sehr schwierig.

28. Eric Raymond ist ein Verfechter der "ego boost" - Theory, zu der er einen weiteren ethnografischen, aber falschen Vergleich zwischen Freier Software und dem Kwakiutl Potlatch hinzufügt. Vgl. *Eric S. Raymond*, 1998. [Homesteading the Noosphere.](#) Aber der Potlatch, sicher eine Form des Statuswettbewerbs unterscheidet sich von Freier Software in zwei wesentlichen Punkten: Er ist essentiell hierarchisch und - was wir wissen, seitdem Thorstein Veblen das erste Mal unsere Aufmerksamkeit darauf gelenkt hat - es ist eine Form der konspirativen Verschwendung. Vgl. *Thorstein Veblen*, 1967. *The Theory of the Leisure Class*. New York: Viking, p. 75. Diese Punkte unterscheiden gerade die anti-hierarchische und utilitaristische Kultur freier Software von ihren proprietären Gegenmodellen.

29. *Vinod Valloppillil*, [Linux OS Competitive Analysis \(Halloween II\).](#) Beachtlich ist auch die Überraschung Valloppillils, daß ein in Kalifornien geschriebenes Programm von einem Programmierer in Ungarn dokumentiert wird.

30. Vgl. "Sie spielen unser Lied: Der Tag, an dem die Musik - Industrie starb" in *Das Unsichtbare Barbecue*, im Erscheinen.

31. *International News Service v. Associated Press*, 248 U.S. 215 (1918). 32. Vgl. "Kein Wunderkind: Die politische Theorie universeller Vernetzung," in *Das Unsichtbare Barbecue*, im Erscheinen.