Till A. Heilmann

Digitale Kodierung und Repräsentation. DVD, CSS, DeCSS

2010

https://doi.org/10.25969/mediarep/675

Veröffentlichungsversion / published version
Zeitschriftenartikel / journal article

Empfohlene Zitierung / Suggested Citation:
Heilmann, Till A.: Digitale Kodierung und Repräsentation. DVD, CSS, DeCSS. In: *Navigationen - Zeitschrift für Medien- und Kulturwissenschaften*, Jg. 10 (2010), Nr. 2, S. 95–112. DOI: https://doi.org/10.25969/mediarep/675.

Erstmalig hier erschienen / Initial publication here:
https://nbn-resolving.org/urn:nbn:de:hbz:467-5698

UNIX, Unix, *nix

Kopierschutz in der Softwareentwicklung

VON ALEXANDER FIRYN

»SCO is claiming parenthood of that child and now
wants to make money off the earnings of that child. Even
though SCO has refused to undergo the technical
equivalent of DNA testing, and even though my
(and other people's) DNA is probably all over Linux.«
(Linus Torvalds, 2003)

TECHNISCHER KOPIERSCHUTZ UND TECHNISCHE STANDARDS

Wenn von technischem Kopierschutz die Rede ist, dann geht es mehrheitlich um den Schutz von Nutzdaten, etwa Text-, Musik- oder Videodaten. Im einfachsten Fall sind die Besitzverhältnisse an den zu schützenden Daten vollkommen klar, der legitime Verwerter hat eine genaue Vorstellung davon, wem er welche Rechte an den Daten zugestehen möchte und die für die Distribution der Daten ausgewählten Kanäle unterstützen die für die Umsetzung dieser Vorstellung notwendigen Maßnahmen. Moderne Verfahren für das *Digital Rights Management* (DRM), wie sie etwa von Microsoft, Apple oder Adobe bereitgestellt werden, erfüllen selbst exotische Wünsche von Verwertern und verwenden in der Zwischenzeit so sichere Verschlüsselungsverfahren, dass bei der Umgehung der eingesetzten Schutzmaßnahmen wohl tatsächlich eine gewisse kriminelle Energie vorausgesetzt werden muss, die zumindest im juristischen Sinne als hinreichender Beleg für einen Straftatbestand gewertet werden darf, der laut UrhG, §95a Abs. 1 in Deutschland dann vorliegt, wenn »wirksame technische Maßnahmen zum Schutz eines nach diesem Gesetz geschützten Werkes [...] ohne Zustimmung des Rechteinhabers [...] umgangen werden«.

Natürlich gibt es auch weniger einfache Fälle. Als Philips und Sony vor fast dreißig Jahren mit der Spezifikation der Audio-CD begannen, dachte von den Entwicklern niemand daran, dass es eines Tages einen Bedarf an technischen Kopierschutzmaßnahmen für das neue Medium geben würde. Die 1980 veröffentlichte ANSI-Spezifikation[1] der Audio-CD, das sog. *Red Book*[2], sah entsprechend einen

1 Das *American National Standards Institute* (ANSI) wurde 1916 von drei amerikanischen Ministerien, dem Kriegsministerium, dem Schifffahrtsministerium und dem Wirtschaftsministerium, gegründet und zählt neben der internationalen Organisation für Normung (ISO) und deren nationalen Vertretungen, etwa der DIN in Deutschland, zu den wichtigsten Standardisierungsgremien für Technologien und Prozesse. Wie in den meisten Standardisierungsgremien kann auch bei der ANSI jedes – in der Regel dafür zahlende –

solchen Schutz nicht vor. Als 20 Jahre später der Bedarf da war, waren weltweit längst Millionen von Audio CD-Playern im Einsatz, die von ihren Entwicklern brav nach ANSI-Spezifikation entwickelt worden waren. Nachträglich einen technisch wirksamen Kopierschutz in die Audio-CD einzubauen, hätte bedeutet, all diese Geräte in nutzlose Wohnungsdekorationen zu verwandeln. Der dabei sicher entstandene Vertrauensverlust der Kunden in eine ganze Industrie hat glücklicherweise eine Erweiterung des *Red Books* um DRM-Verfahren verhindert. Die Musikindustrie versuchte nun, die effizienten Fehlerkorrekturalgorithmen, die zwar in den meisten reinen Softwareplayern, üblicherweise aber nicht in Hardwareplayern, enthalten waren, durch geschickt angeordnete Bitfehler in den Daten der CDs zum Absturz zu bewegen. Das war für kurze Zeit in vielen Fällen sehr verbreiteter Software wirksam, allerdings auch ausgerechnet bei vielen teuren und hochwertigen Hardwareplayern und führte letztlich nur zu einer Generation neuer Software mit angepassten Korrekturverfahren – und einem massiven Protest der Besitzer hochwertiger CD-Player.[3]

Weitsichtiger war die Industrie bei der Entwicklung der Video-DVD, die von vornherein ein Kopierschutzverfahren enthielt. Mit dem *Content Scrambling System* (CSS) können die Daten auf Video-DVDs verschlüsselt werden. Der Algorithmus für die Verschlüsselung ist standardisiert, einer der gültigen Schlüssel wird Herstellern von DVD-Playern in Hard- oder Software gegen eine Lizenzgebühr zur Verfügung gestellt. Der Einfachheit halber wurde allerdings kein Verfahren für den Austausch oder die Sperre eines Schlüssels bereitgestellt. Würde ein gültiger Schlüssel einmal bekannt werden, könnte jeder durchschnittlich begabte Programmierer eine Software zum lesen – und kopieren – von DVDs entwickeln. Und natürlich wurden Schlüssel bekannt und das schon vor elf Jahren, 1999.[4]

Obwohl Audio-CD und Video-DVD de Facto schutzlos ausgeliefert werden, gibt es keinen Grund anzunehmen, Nutzdaten ließen sich nicht hinreichend vor unerwünschten Zugriffen schützen. Beide Formate sind vor Urzeiten entwickelt worden und beide sind durch die massenhafte Verbreitung von Consumer-Endgeräten dazu verdammt, sich bis zu ihrem bitteren Ende an ihre mangelhaften Standards zu halten. Aber Hand aufs Herz: dieses Ende ist längst eingeläutet. Die

---

Mitglied Vorschläge für Standards machen, die in einem aufwändigen Prozess von Experten geprüft, diskutiert, ggf. überarbeitet und im besten Fall verabschiedet werden.

2 Das *Red Book* ist das erste der sog. *Rainbow Books*, in denen die verschiedenen Standards für CD-Formate definiert sind (vgl. Lehtinen/Russell/Gangemi 2006).

3 Der Heise-Verlag, der in Deutschland die weit verbreiteten Computerzeitschriften *iX* und *c't* herausgibt, hat zu dieser Zeit in Deutschland den Begriff der »Un-CD« geprägt und eine umfangreiche Online-Datenbank mit nicht Standardkonformen Audio-CDs aufgebaut. Dass die Musikindustrie diesen Weg aufgegeben hat, wird auch dadurch belegt, dass diese Datenbank ebenso wie ihre diversen internationalen Pendants inzwischen nicht mehr bereitgestellt wird.

4 Auf dem ersten *Hack*, *DeCSS*, basiert noch heute die *libdvdcss*, die bis vor nicht langer Zeit die einzige technische Möglichkeit bot, unter Linux verschlüsselte Video-DVDs anzuschauen (vgl. Schwabach 2006). Vgl. den Beitrag von Till Heilmann in diesem Heft.

DVD verblasst neben BlueRay und stirbt mit jedem verkauften Großbild-Fernseher einen kleinen Tod. Und den iPod oder irgendeinen seiner Verwandten mit Audio-CDs zu füttern, kommt nur noch für die Archivierung längst verstaubter Bestände in Frage. Die neuen Formate bieten mehr Schutz, als die Musikindustrie inzwischen noch für notwendig hält und die Zeit, als in die Erforschung von DRM-Technologien Milliarden investiert wurden, ist vorbei. Die noch aktiv betriebenen Forschungsfelder aus dem Dunstkreis der DRM-Technologien, etwa das *Perceptual Hashing*[5], werden längst auf den Einsatz in verbesserten Suchtechnologien hin optimiert statt für den Einsatz bei der Plagiatssuche. Und im Laufe des vergangenen Jahres haben alle großen Plattenlabels den Onlinevertrieb von Digitalisaten ohne *technisch* erzwungene Nutzungseinschränkungen erlaubt, nachdem die Industrie erkannt hat, dass der allzu sichere Schutz sich negativ auf den Absatz auswirkt.

Diese Beispiele zeigen, dass die Spielräume für die Einführung und Modernisierung *technischer Kopierschutzverfahren* zum einen davon abhängen, ob und in welchem Maße solche Verfahren in den zugrundeliegenden *technischen Standards* der zu schützenden Medien vorgesehen sind, zum anderen davon, wie weit diese technischen Standards verbreitet sind, wie weit sie also als *de facto Standards* tatsächlich zum Einsatz kommen.

Für die oben besprochenen Nutzdaten entscheiden sich die Möglichkeiten der Einführung technischer Schutzverfahren im Wesentlichen im Zuge einer Kosten-/Nutzenrechnung durch die Industrie. Viel komplizierter ist die Lage allerdings, wenn es nicht um den Schutz der Endprodukte für den Endkunden geht, sondern um den Schutz partieller Produktbestandteile im Produktentstehungsprozess. Bevor etwa ein Hollywood-Film auf DVD gepresst an den Endkunden ausgeliefert wird, mussten schon hunderte, oft tausende von Menschen in den verschiedensten Unternehmen Zugriff auf das zugrunde liegende Material haben. Das betrifft die Cutter, die Digital Composer, die Synchronisationsfirmen, die Kopierwerkstätten und viele weitere Instanzen. Wenn nur an einer dieser Stellen ein Leck entsteht, durch das ein vollständiger Film – unter Umständen schon vor der Kinopremiere – an die Öffentlichkeit dringt, geht es nicht mehr um einen Schaden von ein paar hundert Euro durch illegale DVD-Kopien, sondern schnell um Millionenverluste. Was die Einführung technischer Schutzverfahren in diesen Teil des Prozesses so schwierig macht, ist nicht etwa der Kampf gegen de facto Standards, sondern der Kampf gegen die Komplexität des Prozesses. Effiziente Schutzverfahren müssten jeden Schritt der kooperativen Produktentstehung reflektieren und würden schon bei kleinen Fehlern den gesamten Arbeitsprozess massiv behindern. Um diesem diffusen Begriff der Komplexität Fleisch zu geben, sollen die Schwierigkeiten der Realisierung eines technischen Kopierschutzes, der im Pro-

5 *Perceptual Hashing* bezeichnet eine Gruppe technischer Verfahren zur Erzeugung digitaler Wasserzeichen, mit denen nicht nur 1:1-Kopien digitaler Dateien, sondern auch Varianten mit einem bestimmten Maß an Unterschieden zum Original als Kopie identifiziert werden können (vgl. Ng/Nesi 2008).

duktentstehungsprozess vornehmlich digitaler Produkte wirksam werden müsste, an einem Beispiel aufgezeigt werden, nämlich der Mutterdisziplin aller digitalen Produktentwicklungen: der Entwicklung von Software. Konkret wird diese Problematisierung am Beispiel der Entwicklung des Betriebssystems UNIX vollzogen, das wegen seiner langen Geschichte fast als ein Vergrößerungsglas der Herausforderungen gesehen werden kann, die letztlich alle digitalen Produktlebenszyklen treffen.

VERGLEICHE

> »Dear Steve: As you requested below is a draft of my report on existence the of Unix derived code in Linux. What we tried to do is to determine if there was any material from Unix in the Red Hat Linux release 5.2. To make this determination we used a copy of Red Hat Linux which was purposed from the local Best Buy. We then compared it to multiple copies of Unix. We received sources from SCO of OpenServer 5.0 dated January 27, 1998, Gemini Source dated January 27, 1998, it being UnixWare 7, UnixWare 3.2, UnixWare 4.0 and UnixWare 4.2. Additionally we received various versions of files which were not on the release described. To perform this work we unpacked and obtained the sources for Red Hat from the CDs which are provided and files in Linux and the various versions of Unix. For example we compared Unix yacc to the Linux bison and the Unix awk to the Linux mawk. We performed this comparison on all files which had similar functionality.« (Swatz 1999: 1)

Mit diesen Worten aus einem Memorandum an Steve Sabbath, seinerzeit Vice President of Law der Santa Cruz Operation (SCO), beginnt eines der traurigeren Kapitel der Softwaregeschichte. SCO war einst berühmt und erfolgreich geworden, weil sie das – gerade im profitablen Unternehmensgeschäft – wichtige Betriebssystem UNIX in einer Version verkauften, die auf der i386-Architektur lief, also auf handelsüblichen Intel-kompatiblen PCs, die sich auch Privatanwender und eben kleine Unternehmen leisten konnten. Im Laufe der neunziger Jahre konnte SCO in diesem Segment erhebliche Marktanteile gewinnen und verdiente genug Geld, um 1995 sogar den Quellcode von Novells UnixWare erwerben und damit in das Unix-Geschäft auch mit Großrechnern einsteigen zu können.

Damit ist der glückliche Teil der Unternehmensgeschichte von SCO aber auch schon erzählt: Novell konnte sich die Hände reiben, UnixWare für viel Geld verschachert zu haben, bevor der Markt für Unix-Großrechnersysteme einbrach und SCO musste mit ansehen, wie nicht nur das gerade neu betretene Segment sich auflöste, sondern auch die Einnahmen im alten i386-Geschäft wegbrachen. Diese Entwicklung hatte vor allem drei Ursachen: Die i386-Plattform wurde mit Einführung des Pentium-Prozessors 1993 erheblich leistungsfähiger, so dass es sich für immer mehr Unternehmen lohnte, viele preiswerte PCs anzuschaffen,

statt eines (sehr) teuren Großrechners und vieler preiswerter Terminalplätze. Zugleich gelang es Microsoft, sein Betriebssystem Windows auch im Bewusstsein von IT-Entscheidern in Unternehmen mit der Aura eines konkurrenzlosen Standards zu etablieren. Mit Windows NT 4.0 existierte spätestens 1998 eine Windows-Variante, die zumindest die am häufigsten benötigten Funktionalitäten eines Servers unterstützte und zugleich viel preiswerter war als Unix. Und letztlich wurde im Laufe der neunziger Jahre eine ganze Reihe quelloffener Nachbauten von Unix entwickelt, die vollkommen kostenlos über das Internet verteilt wurden und ebenfalls auf i386-PCs liefen. Der erfolgreichste dieser Nachbauten war schon Ende der 1990er Jahre Linux und weil Linux den Markt für kostenpflichtige i386-Unixe vernichtete, war es SCO ein Dorn im Auge.

Noch wenige Jahre vorher war es kaum denkbar, dass der gesamte Unix-Markt eines Tages einbrechen könnte. Die Geschichte von UNIX beginnt schon in den 1960er Jahren, lange vor der aller anderen heute noch bekannten Betriebssysteme. Einige Entwickler der Bell Laboratories, vor allem Ken Thompson und Dennis Ritchie, waren damals in ein großes Entwicklungsprojekt zahlreicher Unternehmen involviert, dessen Ziel es war, ein standardisiertes Betriebssystem zu schaffen, das auf vielen verschiedenen Computern lauffähig sein sollte. Thompson und Ritchie hatten viele Vorschläge für dieses System, das MULTICS heißen sollte, aber in einem so großen Konsortium ließen sich nur wenige dieser Vorschläge durchsetzen. Als die Bell Labs sich 1969 aus dem Konsortium zurückzogen, hatten die beiden die Freiheit, Ihre Pläne intern umzusetzen – und wohl auch den Ehrgeiz, ein besseres Ergebnis zu schaffen als der Rest des Konsortiums, das sich währenddessen weiter über MULTICS stritt.[6] Der Vorzug des nun kleinen Teams waren die kurzen Entscheidungswege, die eine ganze Reihe seinerzeit sehr radikaler Designentscheidungen ermöglichten. In diesem Sinne war es sicherlich auch hilfreich, dass die Bell Labs selber kein strategisches Interesse an diesem Projekt hatten und so zwar wenig Unterstützung leisteten, aber eben auch nicht störend eingriffen. Thompson, Ritchie und einige Kollegen entwickelten eine eigene Programmiersprache, C[7], ein sehr leistungsfähiges Dateisystem und vor allem eine zukunftsweisende Entwicklungsphilosophie. Statt, wie es bis dahin üblich war, die Software möglichst gut auf die Fähigkeiten der Hardware zu optimieren, legte das Team großen Wert darauf, möglichst einfachen, gut lesbaren und damit auch gut wartbaren Quellcode zu schreiben, der so weit von der Hardware abstrahiert, dass er sich auf möglichst vielen Hardwareplattformen ohne Codeänderungen in Binärcode übersetzen lässt. Schnell, so war der propagierte Gedanke, wird die Software von selber, wenn die Hardware der nächsten Generation schneller wird.[8] Wer seinen Code optimiert, der muss für jede neue Hardware neuen Co-

6 Zur Geschichte von MULTICS vgl. Ceruzzi 2003.

7 Die klassische Einführung in C, die nicht nur für angehende Programmierer immer noch interessant ist, stammt von den Erfindern selber (vgl. Kernighan/Ritchie 1988).

8 Das berühmte *Moore's Law* von der jährlichen Verdoppelung der Schaltkreise auf einem Computerchip und dem damit einhergehenden Geschwindigkeitszuwachs wurde von

de schreiben – und das kostet viel mehr Zeit und Geld, als einfach auf die neue Hardware zu warten. Ein ähnliches Vorgehen hatte IBM bereits in den 1960er Jahren bei der Entwicklung des Betriebssystems OS/360 gewählt. OS/360 sollte nicht auf einzelne Computer, die fast in Handarbeit für die Bedürfnisse eines einzigen Kunden gefertigt wurden, zugeschnitten sein, sondern alle Bedürfnisse erfüllen können, die Kunden an ein Betriebssystem stellen könnten. Während IBM diese Kompatibilität aber dadurch sicherstellte, dass sie eine skalierbare Hardwarearchitektur, das System/360[9], entwickelten, das über alle seine Möglichkeiten hinweg vom OS/360 unterstützt wurde, war der Ansatz von UNIX noch radikaler. UNIX sollte grundsätzlich mit möglichst geringem Aufwand auf jede Hardware jeden Herstellers portiert werden können, was überhaupt erst die Geburtsstunde von Software als eigenständigem, von der Hardware unabhängigem, Produkt markiert.[10]

Ein weiterer wesentlicher Leitsatz der UNIX-Philosophie lautet: »Do one thing and do it right.«[11] Dieser unscheinbare Satz ist untrennbar mit dem Namen Douglas McIlroy verbunden, der das sog. *Pipes*-Konzept erfunden hat.[12] Die Idee ist, grundsätzlich nur winzige Mikroanwendungen zu programmieren, die genau eine Aufgabe erfüllen, als Eingabeparameter eine Zeichenfolge erhalten und als Ergebnis auch wieder eine Zeichenfolge ausgeben. Über eine Folge McIlroy'scher Pipes sollen sich die Ausgaben eines Programms dann als Eingabe für ein weiteres Programm verwenden lassen, so dass zur Erledigung einer komplexen Aufgabe eine Pipeline aus sehr einfachen Programmen erzeugt wird. Dieses Pipe-Konzept wurde in UNIX zum ersten Mal umgesetzt und ist bis heute ein wichtiger Grund für die Leistungsfähigkeit aller Unix-artigen Betriebssysteme. Statt beispielsweise ein Programm zu schreiben, das die Anzahl aller Zeilen einer *Datei* ausgibt, in denen ein bestimmtes *Wort* vorkommt, werden für diese ›komplexe‹ Aufgabe zwei Programme verwendet: eines, das alle Zeilen ausgibt, in denen das Wort vorkommt (*grep*) und ein weiteres, das diese Zeilen zählt und die Summe ausgibt (*wc*), verkettet über den Pipeoperator »|«.

Als Befehl liest sich das Beispiel dann: *grep »Wort« < »Datei« | wc*. Um noch ein Beispiel zur Veranschaulichung zu bringen, nehmen wir den Fall an, ein Nutzer möchte alle Jpeg-Dateien in einer Verzeichnisstruktur auf eine CD-Rom brennen.

Gordon Moore erstmals 1965 postuliert. 1975 korrigierte er seine Prognose auf eine Verdoppelung alle zwei Jahre; 2007 sagte er das nahende Ende seines ›Gesetzes‹ voraus (vgl. Lanzerotti 2006).

9 Zum System/360 vgl. Brooks 1995.

10 Wer dies für einen Widerspruch zu einem der berühmtesten Aufsätze über Computergeschichte hält, lese noch einmal: Kittler: »Es gibt keine Software« (1993).

11 Die beste und vollständigste Darstellung dieser Philosophie, deren Leitsätze hier nur tangiert werden, findet sich in: Raymond: *The Art of Unix Programming* (2004).

12 Als Leiter des Computing Techniques Research Department der Bell Labs zählt McIlroy neben Thompson und Ritchie zu den wichtigsten, in historischen Darstellungen aber häufig unterschlagenen Initiatoren hinter UNIX. Er ist übrigens tatsächlich promovierter Philosoph.

Unter Linux und vielen anderen UNIX-artigen Betriebssystemen könnte dafür eine Pipeline aus dem Programm *find* zum Suchen nach Dateien, dem Befehl *mkisofs* zum Erzeugen eines CD-kompatiblen (ISO-9660 konformen) Dateisystems sowie dem Befehl *cdrecord*[13] zum Schreiben eines Datenstroms auf einen CD-Rohling verwendet werden. Das könnte sich dann (etwas vereinfacht) so lesen: *find . -iname *.jpg | mkisofs -Jr | cdrecord -*.

Auch wenn solche Befehlsfolgen dem einen oder anderen spontan nicht sonderlich attraktiv anmuten, wird eines doch hoffentlich nachvollziehbar: der Entwicklungsaufwand für die Unterstützung immer komplexerer und individuellerer Anforderungen lässt sich mit dem Pipe-Konzept erheblich reduzieren. Statt immer wieder einzelnen Programmen die Fähigkeit zu geben, Dateien nach einem bestimmten Muster finden zu können – mal besser, mal schlechter implementiert – wird diese Funktionalität auf einem Unix-System nur ein einziges mal bereitgestellt, nur an dieser Stelle so gut wie möglich implementiert, und alle Prozesse im System können auf diese eine Implementierung zugreifen. Im Pipe-Konzept spiegelt sich aber auch noch einmal der Leitsatz, auf eine Optimierung zu verzichten. Drei Programme hintereinander zu starten, um eine Aufgabe zu erfüllen, kostet natürlich mehr Systemressourcen, als all diese Funktionen in einem einzigen, speziell für diese Anforderung optimierten Programm zusammenzufassen. Dieser Nachteil wird aber dadurch belanglos, dass sich im Pipe-Konzept nach der einmaligen Entwicklung aller wichtigen Basisfunktionen quasi endlos viele komplexe Anforderungen erfüllen lassen, ohne auch nur einen einzigen weiteren Moment in die Entwicklung zu investieren. UNIX opfert also ein wenig der preiswerten Ausführungszeit, um im Gegenzug sehr viel teure Entwicklungszeit einzusparen.[14]

FRÜHE GENEALOGIE VON UNIX: VON OPEN SOURCE ZUM SCHÜTZBAREN EIGENTUM

Die erste, nach heutigem Verständnis eines Unix, vollständige Version wurde 1973 fertiggestellt und erhielt bei den Bell Labs die Versionsnummer 4. UNIX V4 war vollständig in der neuen Sprache C geschrieben und setzte McIlroys Pipe-Konzept mit einer Menge einfacher Tools um. Die Version war zudem durch das neu entwickelte Dateisystem und die Prozesssteuerung mehrbenutzerfähig – auch eine Eigenschaft, die vollkommen neu war. Diese Version wurde am Anfang nur von den Bell Labs selber genutzt, zunächst für die Textverarbeitungssysteme der eigenen Patentabteilung. Weil es der Muttergesellschaft der Bell Labs, AT&T, seit 1956 verboten war, außerhalb des Telefonmarktes kommerziellen Aktivitäten

13 Weder *cdrecord* noch *mkisofs* zählen zu den historischen UNIX-Programmen, sondern zu den *cdrtools*, die von Jörg Schilling seit 1996 für Linux und viele andere Unix-Derivate entwickelt werden.

14 Gerade unter diesem Aspekt werden die Grundlagen für die bestmögliche Softwareentwicklung auch heute noch in der Tradition der UNIX-Philosophie weiter erforscht; inzwischen unter dem Stichwort »Agile Software« (vgl. etwa Shore/Warden 2007).

nachzugehen, konnten die Bell Labs allerdings sehr schnell anfangen, offensive Werbung für ihre neue Errungenschaft zu machen. Und offensiv hieß hierbei, dass sie UNIX inklusive der Quellcodes jedem unentgeltlich zur Verfügung stellten, der daran ein Interesse hatte.[15] Im Laufe der siebziger Jahre konnte UNIX sich deshalb mit ungeheurer Geschwindigkeit verbreiten und avancierte zum Standardlehrmaterial für Betriebssystem-Vorlesungen an Universitäten. Es wurde aber nicht nur verbreitet: weil die in C geschriebenen Quellcodes ohne Restriktionen mitgeliefert wurden, konnte auch jeder das System nach Belieben ändern und erweitern. Bereits nach wenigen Jahren brachte die University of Berkeley mit der *Berkeley Software Distribution* (BSD) ein eigenständiges System auf den Markt, das sich an vielen Stellen vom Original unterschied – teilweise waren neue Programme hinzugefügt, teilweise waren auch in Bells UNIX vorhandene Programme neu implementiert oder die vorhandenen Implementierungen geändert worden. Bis in die achtziger Jahre hinein war BSD allerdings das einzige UNIX-Derivat, das sich in großen Teilen von der Originalversion unterschied.

Das änderte sich grundlegend, als AT&T sich 1982 durch das Abstoßen einer Reihe von Tochterunternehmen von den Auflagen befreien konnte, die 1956 erlassen worden waren. UNIX war zu diesem Zeitpunkt schon das am weitesten verbreitete Betriebssystem der Welt und der Computermarkt wurde zunehmend interessanter, wenn man Geld verdienen wollte. AT&T erklärte kurzerhand, dass das 1979 veröffentlichte UNIX V7 die letzte frei zur Verfügung gestellte Version bleiben würde und eine künftige Verwendung des UNIX-Codes lizensierungspflichtig sei. Der Schwerpunkt war nun nicht mehr die Entwicklung im akademischen Eigeninteresse, sondern ein kommerzieller Vertrieb. Unter Rückgriff auf eine bereits 1978 bei den Bell Labs begonnene Erweiterung von UNIX, die aber bis dahin unveröffentlicht geblieben war, wurde das Betriebssystem in sehr schneller Zeit um viele Funktionalitäten erweitert, die im frei verfügbaren UNIX V7 nicht enthalten waren.[16] Dieses neue System sollte vor allem als hauseigenes Betriebssystem verkauft werden und zwar ohne eine gleichzeitige Auslieferung des Quellcodes. Weil AT&T klar war, dass auch weiterhin Interesse am Quellcode bestehen würde, den vor allem viele Hardwarehersteller benötigten, um UNIX auf ihre Hardwarearchitekturen zu portieren, wurde 1983 außerdem das UNIX System V freigegeben, das im Unterschied zu den UNIX-Versionen bis V7 allerdings kostenpflichtig lizensiert werden musste. Die Lizenznehmer durften den entstandenen Quellcode auch nicht weitergeben.

Allerdings hatte AT&T die Rechnung ohne die Berkeley University gemacht. In BSD war bis zur Kommerzialisierung des Ur-UNIX bereits viel Geld und Arbeit

15 Nicht ganz kostenlos: Die Interessenten mussten die Kosten für die Datenträger und das Porto übernehmen. Marc Shuttleworth hat auch darauf noch verzichtet, um seine Linux-Distribution Ubuntu seit 2004 als Standard-Linux-Distribution zu etablieren – mit Erfolg.

16 Die Rede ist von *PWB*. Eine sehr wertvolle und stetig aktualisierte graphische Darstellung der gegenseitigen Einflüsse diverser UNIX, Unix und *nix-Derivate stellt Éric Lévénez auf seiner Webseite http://www.levenez.com/ bereit.

geflossen. Unter anderem verfügte BSD bereits 1981 über eine Implementierung des TCP/IP-Protokolls[17], das sich in kurzer Zeit zum grundlegenden Netzwerkprotokoll für das Internet entwickeln sollte. Neben der *Digital Equipment Corporation* (DEC), die mit ihren PDP und VAX-Maschinen zu den wichtigsten Computerlieferanten dieser Zeit gehörte, war auch die DARPA[18], die Forschungsabteilung des amerikanischen Verteidigungsministeriums, Kunde und Förderer der Distribution aus Berkeley. Als dann noch der schnell wachsende Hardwarehersteller SUN auf Basis von BSD das kommerzielle SunOS entwickelte und zu verkaufen begann, war der Geschäftsplan von AT&T erheblich gefährdet. Zum Glück für den erfahrenen Monopolisten wurden in BSD – und damit auch in SunOS – trotz aller Eigenständigkeit immer noch einige wenige, aber wesentliche, Stücke aus dem originalen UNIX-System verwendet, was AT&T Gelegenheit bot, von jedem Kunden von BSD und SunOS Lizenzgebühren für die Nutzung dieser Bestandteile einzufordern. Diese Forderung führte gleichzeitig zu einer Zersplitterung wie auch einer Konsolidierung: Sun und AT&T konnten sich schnell darauf einigen, dass sie mit einem gemeinsamen Vorgehen die besten Chancen im Markt haben würden. Bereits 1986 wurde mit SunOS 3.0 ein grundlegend überarbeitetes Betriebssystem veröffentlicht, das nun nicht mehr auf BSD, sondern auf AT&Ts System V basierte, allerdings die immer noch frei verfügbaren ›mehrwertigen‹ Tools aus BSD integrierte. Gleichzeitig gab die Berkeley University eine Variante von BSD frei, in der keine Komponenten aus dem Besitz von AT&T mehr enthalten waren. Dieses sog. *Networking Release* war zwar ohne den AT&T-Anteil zunächst kein vollständiges, lauffähiges Betriebssystem, enthielt aber soviel Funktionalität, dass die nun fehlenden Teile nach einiger Zeit von neuen Entwicklern nachgerüstet werden konnten. In dieser Linie war 386BSD das erste vollständige Unix-Betriebssystem, das keinen Code des Ur-UNIX mehr verwendete. Die bekanntesten heute noch existierenden Systeme auf dieser Basis sind FreeBSD und das darauf aufbauende Darwin, das wiederum den Kern von Apples Mac OS X bildet.[19]

---

17 Entwickelt wurde die TCP/IP-Familie seit etwa 1972 bei der DARPA von Robert E. Kahn und Vinton Cerf. 1982 wurde TCP/IP zum verbindlichen Standard für militärische Netzwerke in den USA erhoben, was den Vorsprung von BSD gegenüber AT&T-UNIX überhaupt erst zu einem, zumindest kleinen, Wettbewerbsvorteil werden ließ. Die Besonderheiten der BSD-Implementierung sowie die Auswirkungen dieser Implementierung werden in Stevens: *TCP/IP Illustrated* (1995) behandelt.

18 Die *Defense Advanced Research Projects Agency* (DARPA) wurde Ende der 1950er Jahre gegründet, um den amerikanischen Technologievorsprung vor der UDSSR sicherzustellen. Die Behörde hat mit ihren Forschungsprojekten maßgebliche technische und methodische Grundlagen für alle Bereiche der IT-Industrie gefördert (vgl. Ceruzzi 2003; Flamm 1988).

19 Neben FreeBSD bildet eine Weiterentwicklung des Mach-Kernels einen wesentlichen Baustein für Darwin, den Steve Jobs schon Ende der 1980er Jahre als Grundlage für sein (erfolgloses) NeXTSTEP-Betriebssystem gewählt hatte. Der Mach-Kernel selber ist aus 4.2BSD an der Carnegie Mellon University entstanden und war eines der ersten wichti-

UNIX-KRIEG UND KONSOLIDIERUNG

Neben den beiden Lagern, BSD auf der einen Seite, AT&T und Sun auf der anderen, traten noch eine Reihe von Unternehmen auf den Plan, die auf Basis des lizensierten AT&T Quellcodes eigene UNIX-Derivate entwickelten und binär vertrieben. Dazu zählten vor allem Hewlett & Packard mit HP-UX sowie Microsoft, die ihr XENIX aber schon 1984 verkauften: an SCO.[20] Die schnell entstandene Vielfalt an UNIX-Derivaten war aber all den Unternehmen ein Dorn im Auge, die UNIX zwar an Endkunden verkauften, selber aber keine Betriebssysteme entwickelten. Je mehr sich die verschiedenen Derivate voneinander entfernten, desto schwieriger wurde es plötzlich wieder, vorherzusagen, ob eine bestimmte Software auf einem bestimmten UNIX-Derivat laufen wird. Genau diese Sicherheit verlangen aber die Endkunden. 1984 schloss sich eine Gruppe solcher Unternehmen zum X/Open-Konsortium zusammen und formulierte eine Reihe von Standards, die ein Unix-System ihrer Auffassung nach erfüllen müsste, um eine minimale Kompatibilität unter den Derivaten sicherzustellen. Als klar wurde, dass eine solche Forderung wenig Eindruck machte, insbesondere auf AT&T, gründeten einige der X/Open-Mitglieder noch eine zweite Initiative: die *Open Software Foundation* (OSF).[21] Statt nur zu standardisieren, setzten die Mitglieder der OSF die formulierten Standards auch gleich um, indem sie die freien Teile von BSD um eigene Komponenten ergänzten, insb. um einen eigenen Betriebssystemkernel, den *Mach*-Kernel. Aus dieser Initiative entwickelten sich mit NeXTSTEP (1988) und OSF/1 (1990) zwei Betriebssysteme, die die Anforderungen der X/Open erfüllten und keinen Code von AT&T enthielten. Weil auch Hewlett & Packard und IBM der OSF beigetreten waren, wurden auch deren Betriebssysteme HP-UX und AIX vom System V unabhängig, so dass AT&T auch von diesen wichtigen Betriebssystemherstellern keine Lizenzgebühren mehr einfordern konnte. AT&T machte zwar noch einige erfolgreiche juristische Vorstöße wegen Markenrechtsverletzungen gegen X/Open, gegen den Vertrieb der Software konnte AT&T aber keine Rechte geltend machen. Als sich abzeichnete, dass Systeme, die sich an mehrheitlich vereinbarte Industriestandards halten, leichter verkauft werden konnten, trat AT&T schließlich selber dem Konsortium bei. Kurze Zeit später, 1993, verkaufte AT&T sogar seine gesamten UNIX System Laboratories, inklusive dem nun schon erheblich entwerteten UNIX System V, an den Netzwerkspezialisten Novell und konzentrierte seine Gewinninteressen auf den Vertrieb der Markenrechte.

---

gen Ergebnisse der OSF: ein Unix-Kernel, der ohne Codebestandteile von AT&T auskam.

20 XENIX war der erste Versuch von Microsoft, ein Standard-Betriebssystem für IBM-PCs zu liefern. Allerdings war die UNIX-Portierung zu ressourcenhungrig für die frühen IBM-PCs, so dass erst der zweite Versuch, das erheblich leistungsärmere MS-DOS, zum Ziel führte. Mit der Etablierung von MS-DOS als Standardbetriebssystem für PCs war XENIX für Microsoft strategisch wertlos geworden (vgl. Clukey 1985).

21 1996 sind OSF und X/Open zur Open Group fusioniert.

TECHNISCHE SCHUTZMAßNAHMEN

Nach soviel Geschichtsaufarbeitung muss begonnen werden, über Schutzmaßnahmen nachzudenken. Es wird klar geworden sein, dass UNIX eine ganze Reihe von Elementen mitbringt, die aus jeweils verschiedenen Perspektiven des Schutzes bedürfen. Zumindest drei dieser Elemente sollen hier benannt werden:

Zum ersten die Philosophie hinter Unix. Die Unix-Philosophie und ihre konsequente Anwendung bildet die Grundlage für den einmaligen Erfolg dieses Betriebssystems. Als Philosophie allerdings lässt sie sich schwerlich mit technischen Mitteln schützen, sondern nur durch ihre Anwendung. Es wird aber klar werden, dass sie gerade durch ihre Anwendung Einfluss auf die technische Schützbarkeit anderer Elemente hat.

Zum zweiten stellt die Marke UNIX ein schützenswertes Gut dar, das allerdings auch wiederum nicht unmittelbar durch technische Mittel geschützt werden kann.

Zum dritten bedarf die Architektur von Unix, gewissermaßen die Gemeinsamkeit aller verschiedenen Unix-Derivate, eines Schutzes, durch den die Kompatibilität der einzelnen Derivate untereinander auf jeweils verschiedenen Ebenen sichergestellt wird. Dieser Schutzbedarf wird nicht jedem sofort als ein Bedarf an Kopierschutzmechanismen plausibel sein – tatsächlich geht es aber genau darum. In den Anfangsjahren von UNIX schützte die kleine Entwicklergruppe an den Bell Labs die Architektur von Unix dadurch, dass ihr eigenes UNIX die Ausformung dieser Architektur bildete. Technisch gesprochen: Das Betriebssystem UNIX war die einzige Instanz der Unix-Architektur, die unabhängig von dieser Instanz noch überhaupt nicht existierte. Mit dem Beginn der aktiven Eigenentwicklung BSD in Berkeley bildete sich eine zweite Instanz heraus, die sich in vielen Punkten von der Bell Labs-Instanz unterschied, sehr wohl aber auf ihr aufbaute und von ihr profitierte. Die Bell Labs hatten zum Beispiel entschieden, dass sämtliche Netzwerkkommunikation unter UNIX mittels *Streams* realisiert wird, einem speziellen Verfahren, um den Datenaustausch zwischen Betriebssystemkernel und Endanwendungen zu unterstützen. Das war eine Architekturentscheidung die letztlich in jeder Netzwerkanwendung für UNIX reflektiert werden musste. In Berkeley entschied man sich aber gegen Streams und stellte die Netzwerkfunktionen stattdessen über Sockets bereit – mit der Folge, dass für UNIX entwickelte Netzwerkprogramme, etwa FTP, Telnet, später die ersten Webbrowser, entweder nur auf einer der beiden Architekturinstanzen lauffähig waren oder beide Varianten unterstützen mussten, was die Entwicklung jeder einzelnen Anwendung aufwändiger machen müsste. Weil BSD sehr schwergewichtige Förderer hatte, konnte aus Berkeley schon an dieser einen Stelle ein erheblicher Druck auf die Architekturentscheidungen der Bell Labs ausgeübt werden, mit den Anwendungsentwicklern und Endnutzern als Leidtragenden. Bei einer stetig wachsenden Zahl von Instanzen müssen solche Machtkämpfe letztlich zur Auflösung der ganzen Architektur führen.

Dass inzwischen zwar kaum noch ein relevantes Betriebssystem existiert, das in direkter Linie vom ursprünglichen UNIX abstammt[22], dafür aber eine Familie von Unix-Betriebssystemen, die sich durch die Bindung an gemeinsame Architekturentscheidungen auszeichnen und dadurch untereinander – auf verschiedenen Ebenen – kompatibel sind, ist dem technischen Schutz dieser Architektur durch technische Standards zu verdanken, wie sie zuerst vom X/Open-Konsortium gefordert und aufgestellt und dann von der OSF in Instanzen dieser standardisierten Architektur umgesetzt worden sind. Der *X/Open Portability Guide*, der in mehreren Abschnitten von 1984 bis 1992 veröffentlicht wurde, war in seiner letzten Fassung in drei Themenkomplexe unterteilt: Erstens: »System Interfaces and Headers«, also alle technischen Voraussetzungen, die auf einer Instanz der Unix-Architektur gegeben sein müssen, um auf diesen aufbauend Anwendungen entwickeln zu können. Zweitens: »Commands and Utilities«, also alle Anwendungen, die auf einer solchen Instanz mindestens vorhanden sein müssen, inklusive einer Beschreibung der zulässigen Parameter, der Namen der Programme sowie der zu liefernden Ergebnisausgaben – all das ist elementar wichtig, um dieselben Pipelines auf verschiedenen Unix-Instanzen ausführen zu können. Drittens: »System Interface Definitions«, also alle Funktionsaufrufe (Calls), die von Programmen an die Basis des Betriebssystems gemacht werden können müssen sowie die zu liefernden Rückgaben dieser Systemaufrufe. In diesen Bereich fällt beispielsweise auch, dass eine Unix-Instanz Calls bereitstellen muss, die es Netzwerkanwendungen erlauben, über Streams mit dem Betriebssystemkernel zu kommunizieren, der wiederum die reale Hardware, etwa eine Netzwerkkarte, verwaltet. Diese X/Open-Standards bilden auch heute noch die Grundlage für die sog. »Single UNIX Specification«, die zuletzt 2001 von der heute zuständigen Austin Group[23] in Version 3 freigegeben wurde. Jedes Betriebssystem, das heute den Markennamen UNIX nutzen will, muss diese Spezifikation in der aktuellen Form fehlerfrei umsetzen. Ergänzt wird dieser Standard durch den sog. POSIX-Standard[24], in dem noch viele weitere Gebiete behandelt werden, auf denen technische Kompatibilität wichtig ist, dabei aber keine grundsätzliche UNIX-Kompatibilität fordert. Der POSIX-

22 SUN lässt sein Solaris seit 2004 überwiegend unter einer Open Source Lizenz – und unter dem Namen *Open Solaris* - entwickeln. In diesem Zuge werden die historischen Bestandteile aus UNIX Stück für Stück ersetzt. Bleibt als verbreitetes System aus der UNIX-Linie noch HP-UX, dessen Verbreitung sich aber auf einen kleinen Teil der von HP vertriebenen Server beschränkt und das auf den inzwischen wichtigsten Hardwareplattformen, i386 und x86_64, nicht lauffähig ist.

23 Die Austin Group ist eine Arbeitsgruppe innerhalb der Open Group. Auf der Webseite der Austin Group werden zahlreiche Protokolle zur Verfügung gestellt, die einen interessanten Einblick in die Arbeit eines Standardisierungsgremiums geben.

24 Der Name POSIX wurde vom Leiter des GNU-Projekts, Richard Stallman, eingeführt und steht für *Portable Operating System Interface*. Der offizielle Name des Standards, der vom *Institute of Electrical and Electronics Engineers* (IEEE) verabschiedet wird, lautet »IEEE 1003.1«. IEEE 1003.1 ist zugleich von der ISO als »ISO9945« bestätigt worden (vgl. dazu die UNIX-Seiten der Open Group unter www.unix.org).

Standard lässt sich also auch erfüllen, ohne Pipe-Konzept und viele andere wesentliche Teile der Unix-Philosophie und der UNIX-Architektur umzusetzen. Insbesondere im Themenkomplex der Netzwerkkommunikation werden in POSIX stattdessen viele umfangreichere und präzisere Anforderungen an POSIX-konforme Betriebssysteme gestellt, deren Einhaltung den Datenaustausch zwischen verschiedenen Betriebssystemen sicherstellt. So verfügt etwa auch Microsofts Betriebssystem Windows über optional installierbare POSIX-Erweiterungen, so dass es zu den POSIX-konformen Systemen gehört – ohne auch nur im Ansatz ein Unix zu sein.

Jeder, der in der Lage ist, einen Standard mitzubestimmen oder sogar zu kontrollieren, der sich als technischer Standard in einer ganzen Industrie durchsetzen lässt, hat viel mehr Macht als jene, die nur über Produkte verfügen, die sich an diese Standards halten. Insofern war es eine brillante Entscheidung von AT&T, sich aus der Entwicklung einer einzigen Instanz von Unix – UNIX – zurückzuziehen, um stattdessen zumindest einen Teil der Kontrolle über alle Instanzen der Unix-Architektur zu erringen. Wer diese sehr effektive Art des Kopierschutzes ›knacken‹ will, der muss aus eigener Kraft einen neuen Standard setzen, den er selber kontrollieren kann. Das aber ist eine schwierige Aufgabe.

Obwohl mit dem Aufkommen der Macht von Architekturen über die konkrete Implementierung einzelner Instanzen der konkrete Quellcode einer Implementierung fast in den Hintergrund tritt, steckt doch auch in der Implementierungsarbeit soviel Zeit, Wissen und letztlich Geld, dass dieser Quellcode als viertes Element mit Schutzbedarf zu bemerken ist. Nun lässt sich Quellcode am effektivsten vor unerwünschten Kopien schützen, indem man ihn gar nicht erst verfügbar macht. Auch hier helfen die oben benannten technischen Standards, den Quellcode zu schützen, ohne den Wert des Systems zu gefährden. Wer eine Anwendung für eine bestehende Instanz der Unix-Architektur entwickeln will, kann zwar die notwendigen Informationen über die verfügbaren Schnittstellen, die Ein- und Ausgabemöglichkeiten, die bereitstehenden Systemcalls etc. aus dem Quellcode dieser Instanz extrahieren – so ihm dieser zur Verfügung steht –, das wäre aber ohnehin eine mühselige Aufgabe. Viel effizienter ist es, wenn er sich bei seiner Entwicklung nicht an der zugrundeliegenden Instanz orientiert, sondern an den übergeordneten Architektur-Standards, die wohl dokumentiert zur Verfügung stehen. Für den Hersteller des Betriebssystems entfällt damit die Notwendigkeit, einem Anwendungsentwickler den eigenen Quellcode zur Verfügung zu stellen. Der Anwendungsentwickler profitiert davon, dass seine Anwendung nicht nur auf der einen Instanz laufen wird, sondern auf allen Instanzen der Architektur, auf deren technischen Standards er aufbaut. Das heißt, auch für Quellcodes können technische Standards als Kopierschutzverfahren herangezogen werden.[25]

25 Das ist sogar der übliche Weg, Schnittstellen für die Softwareentwicklung bereitzustellen.

Eine Ausnahme muss hier allerdings gemacht werden und die führt auf das Memorandum an Steve Sabbath zurück: Wer nicht nur Anwendungen für Unix-Instanzen entwickeln will, sondern selber eine vollständige eigene Instanz von Unix kontrollieren möchte, der wird vom Einblick in den Quellcode einer anderen Unix-Instanz unter Umständen profitieren können.

SCO VS. RED HAT

Novell erkannte recht schnell, dass der Abkauf des Ur-UNIX von AT&T dem Unternehmen kaum Nutzen bringen würde, weil der Markt für UNIX einzubrechen begann und die Entwicklung guter Unix-Systeme vom originalen UNIX-Code kaum noch profitieren konnte. Bereits zwei Jahre nach dem Kauf, 1995, verkaufte das Unternehmen seine UNIX-Abteilung und die bei Novell entwickelten, auf UNIX basierenden, Derivate UnixWare und OpenServer weiter an SCO. SCO hielt damit zwar die Rechte am Quellcode dieser Derivate, war aber in die wichtigen Gremien, die die Architektur kontrollierten, nicht involviert. Als man endlich auch bei SCO merkte, dass diese Derivate keinen nachhaltigen Wert mehr hatten, wurde ein Schuldiger gesucht – und in Form des jungen, aber extrem erfolgreichen Unternehmens Red Hat schnell gefunden, vielleicht zu schnell.

Red Hats Erfolg basierte auf einem Betriebssystem, das Unix in vielen Punkten sehr ähnlich war, selber aber nicht auf Unix basierte, nämlich *Linux*. Die Entwicklung von Linux wurde Anfang der neunziger Jahre von einem einzigen Studenten, Linus Torvalds, begonnen, der zunächst nur zum Eigenbedarf ein sehr einfaches System entwickelt hatte, mit dem er von seinem heimischen PC auf die Unix-Großrechner in seiner Universität zugreifen konnte. Als Torvalds in einer UseNet-Group im Internet nach einer Bezugsquelle für den POSIX-Standard fragte, den er übungshalber bei seiner Entwicklung berücksichtigen wollte, wurden einige Leser aufmerksam und boten an, Torvalds bei seiner Arbeit zu unterstützen.[26] In relativ kurzer Zeit entstand so tatsächlich ein kleiner Betriebssystemkernel, der einige POSIX-Anforderungen erfüllte und seit der ersten veröffentlichten Version Linux[27] genannt wurde. Bereits vor dieser Entwicklung, 1984, hatte Richard Stallmann, ein früherer Entwickler am MIT, aus Protest gegen die Kommerzialisierung von UNIX seinen Arbeitsvertrag gekündigt und das GNU-Projekt gegründet. GNU, als Akronym für *GNU is not UNIX*, verfolgte das Ziel, ein vollständiges Betriebssystem zu entwickeln, das die UNIX-Philosophie umsetzte, auch den POSIX-Standard, nicht aber zwingend die X/Open-Standards. Vor allem

26 Dass Torvalds für diese Diskussionen und die dann folgende Koordination der Entwicklung die Newsgroup des *NIX-Systems Minix des berühmten Informatik-Professors Andrew S. Tanenbaum nutzte, führte schon recht früh zu legendären Auseinandersetzungen zwischen Torvalds und Tanenbaum (vgl. Torvalds/Daimond 2002).

27 Torvalds selber hatte als Namen FREAX vorgeschlagen. Zum Glück hat der Administrator des FTP-Servers eine sehr eigenmächtige Entscheidung gegen Torvalds Vorschlag gefällt.

aber sollte jedes noch so kleine Stückchen Code des GNU-Systems bis in alle Ewigkeit Open Source bleiben, der Quellcode also jedem legitimen Nutzer zur Verfügung gestellt werden, inklusive dem Recht, diesen Code nach Belieben zu ändern. Einzige Einschränkung war, dass auch dieser geänderte Code wieder unter denselben Bedingungen weitergegeben werden musste. Quellcode unter der GNU-Lizenz darf also niemals ›geschlossen‹ werden.[28] Das GNU-Projekt entwickelte in kurzer Zeit viele der Anwendungen, die ein vollständiges System ausmachten, nicht aber einen Betriebssystemkernel.[29] So lag es für Torvalds nah, die GNU-Anwendungen mit seinem Kernel zusammenzuführen und so ein vollständiges Betriebssystem, sowohl in der Hardwareunterstützung wie auch auf der Anwendungsebene, zu haben. Und weil auch Torvalds keine kommerziellen Interessen verfolgte, lizensierte er seinen Kernel unter der selben Lizenz, die auch das GNU-Projekt verwendete.[30]

Weil weder Linux noch GNU einen zwingenden Grund hatten, sich an die standardisierte Unix-Architektur zu halten, hatten sie bei der Entwicklung erhebliche Freiheiten, ein moderneres System zu schaffen. Technische Standards sichern zwar die Kompatibilität, sie zementieren aber eben auch Designentscheidungen, die Entwicklungsfortschritte behindern. Viele der bewussten Inkompatibilitäten von GNU/Linux machten das System tatsächlich effektiver nutzbar, als das beim großen Vorbild der Fall war. Beispielsweise existierte in UNIX schon sehr früh das Programm *grep*, mit dem in Textdateien nach Zeichenketten gesucht werden konnte. Dieses POSIX-grep konnte aber immer nur Dateien in einem Verzeichnis durchsuchen, was durchaus im Sinne der UNIX-Philosophie war. Wenn man also unter UNIX eine ganze Verzeichnishierarchie nach Dateien durchsuchen will, die eine bestimmte Zeichenkette enthalten, muss man eine Pipeline aus *grep* und *find* verwenden, etwa: *find . | grep »Zeichenkette«*. Das grep, das GNU bereitstellte, beherrschte selber die Möglichkeit, rekursiv durch Verzeichnisbäume zu gehen, so dass ein einfaches *grep -r »Zeichenkette«* ausreichte. Das war nicht ganz die reine Schule der UNIX-Philosophie und deshalb in der Unix-Architektur auch anders gefordert. Es war aber ungemein praktisch und fand deshalb viele Liebhaber. Neben dem in Versalien geschriebenen *UNIX* und den nicht auf den Quellcode von UNIX zurückgehenden – nun klein geschriebenen – Unix-Derivaten bildet Linux den prominentesten Vertreter der *nix-

28 Gemeint ist die *GNU General Public License* (GPL), unter deren zweiter Version (GPLv2) Linux lizensiert ist.

29 Eine erste Version des GNU-Betriebssystemkernels *Hurd* erschien bereits 1991, produktiv nutzbar ist Hurd aber bis heute nicht. Weitere Informationen zu Hurd gibt es auf der Webseite des GNU-Projekts.

30 Wobei Torvalds die Lizenz vor allem verwendete, weil er sich Vorteile für die technische Entwicklung versprach, während das GNU-Projekt mit der Lizenz politische Ziele verfolgt. Die Version 3 der GPL, in der Stallmann die politischen Ziele noch weiter in den Vordergrund gestellt hat und dafür auch Behinderungen der technischen Entwicklungsspielräume in Kauf genommen hat, wird von Torvalds deshalb bis heute abgelehnt (vgl. Schäfer 2007).

Betriebssysteme, die sich zwar an UNIX und Unix anlehnen, selber aber kein Unix sind.[31]

Red Hat gründete sich 1993 als eines der ersten Unternehmen, die mit diesem sehr praktischen Betriebssystem Geld verdienen wollten. Dazu stellte Red Hat eine Auswahl der GNU-Anwendungen mit einer gut getesteten Version des Linux-Kernels zusammen, trug ein selbstentwickeltes Installationsprogramm und einige Wartungstools[32] bei und lieferte dieses Paket als vollständige Distribution aus, inklusive Support für jene, die auch bei der Verwendung dieser fertigen Zusammenstellung noch Unterstützung brauchten. Weil die ganze verwendete Software kostenlos im Internet zur Verfügung stand und Red Hat so auch für die installationsfreundliche Zusammenstellung nur wenig Geld nehmen konnte[33], wurde das Unternehmen von den alteingesessenen Unix-Firmen ebenso verlacht wie auch alle anderen Linux-Distributoren. Das änderte sich, als Red Hat Jahr für Jahr größere Gewinne mit dem Support des quasi verschenkten Systems verbuchen konnte, während die Gewinne durch den Vertrieb von Unix-Lizenzen immer weiter einbrachen. Als Red Hat im August 1999 an die Börse ging, gelang eine fast unglaubliche Kapitalisierung von 3,48 Milliarden Dollar – und diese Bewertung musste in der folgenden Dotcom-Blase nicht einmal nach unten korrigiert werden. Zum Vergleich: Novell hatte 1993 gerade einmal 350 Millionen Dollar für sämtliche UNIX-Rechte samt der UNIX System Laboratories an AT&T gezahlt und seitdem hatte dieses Paket einen stetigen Wertverlust erlitten.

SCO schwammen die Felle davon und ihr Vice President Sabbath beauftragte umgehend einen externen Consultant, Robert Swartz, Beweise dafür zu finden, dass in der Red Hat Distribution unerlaubt Quellcode von UNIX eingesetzt wurde.

Robert Swartz stand nun vor einer großen Herausforderung. Er hatte zwar Zugriff auf die verschiedenen Versionen des UNIX-Quellcodes, die ja im Besitz von SCO waren, auch auf den Red Hat-Quellcode, der wegen der GNU-Lizenz frei verfügbar war. Insgesamt standen ihm damit aber schätzungsweise an die hundert Millionen Zeilen Quellcode zur Verfügung[34], in denen er nach übereinstimmenden Zeilen zu suchen hatte.

---

31 Der bekannteste Vertreter dieser Gattung neben Linux ist Minix, das von Andrew S. Tanenbaum seit 1984 als Lehrbetriebssystem entwickelt wird und unter dem Torvalds die ersten Versionen von Linux entwickelt hat (vgl. Torvalds/Daimond 2002).

32 Das bekannteste und einflussreichste Tool ist der *Red Hat Package Manager* (RPM). Bis heute verwenden die meisten Linux-Distributionen das RPM-Format, um die System- und Anwendungssoftware zu verwalten.

33 Dass die wichtigste Einnahmequelle im IT-Geschäft einmal im Beratungs- und Support-Bereich liegen würde und nicht im Lizenzgeschäft, mussten viele IT-Riesen in den letzten Jahren sehr schmerzhaft lernen (vgl. Stare/Rubalcaba 2008).

34 Das ist eine sehr grobe Schätzung. Dokumentiert ist lediglich, dass der Linux-Kernel 1999 aus ca. 2 Mio. Zeilen Quellcode bestand. Der Kernel macht aber nur einen Bruchteil der gesamten Distribution aus, die Swartz untersucht hat. Eingeflossen sind auch die

Swartz beschreibt in seinem Memorandum an Sabbath ein systematisches Vorgehen:

> »We used the following method to determine whether there was any similarity between the Linux and the various releases of Unix. First we found the comparable files in the various version of Unix and Red Hat. This might not always be files with the same name. Further in general for the purposes of this work we would often concatenate the files together which represented a single programm. We then used a programm call ef to perform the comparison.
>
> Ef works by looking for the number of consecutive line in two files which are identical. So for example if you have two files A and B.

a	*K*
b	*L*
c	*M*
d	*duplicate1*
duplicate1	*duplicate2*
duplicate2	*X*
e	*Y*
f	*Z*

> Then the program ef would report that the lines, duplicate1 and duplicate2 where in both files. The program ef can detect similarities as small as one line.« (Swartz 1999: 2)

Swartz' Verfahren mag auf den ersten Blick etwas naiv anmuten, es dürfte aber tatsächlich auch heute noch der in der Praxis einzig gangbare Weg sein, Code-Plagiate aufzuspüren. Zugleich ist das Verfahren damit das einzig praktikable technische Verfahren, um den Kopierschutz von Quellcode in komplexen Softwareprojekten zu unterstützen. Um dieses Ziel zu erreichen, muss die Existenz kopierter Codefragmente erst einmal nachgewiesen werden. Diesen Weg manuell zu gehen, ist bei Millionen von Codezeilen in zigtausenden von Quelldateien schlechterdings unmöglich. Und weil Programmiersprachen im Unterschied zu natürlichen Sprachen einen sehr kleinen Sprachschatz und kaum Möglichkeiten

Libraries und ein Teil der Anwendungen – soweit sie Vorläufer im Ur-Unix hatten. Relevant dürften ca. 20 Mio. Zeilen Code der Red Hat-Distribution gewesen sein, die Swartz laut seiner Beschreibung mit fünf verschiedenen UNIX-Derivaten verglichen hat.

zur spontanen Wortschöpfung mitbringen, würde auch die Suche nach exotischen Wortschöpfungen kaum fruchten.

Hätte Swartz nun nach Codebestandteilen aus Microsoft Windows, aus Mac-OS Classic oder irgendeinem anderen proprietären Betriebssystem suchen sollen, wäre das Verfahren wohl effizient gewesen, in dem Sinne, dass dem Vergleich eine gewisse Aussagekraft zugekommen wäre. Aber es ging um UNIX und damit um eine standardisierte Betriebssystemarchitektur, deren maßgebliche Designrichtlinien seit X/Open nicht mehr einfach aus dem Quellcode hervorgingen, sondern öffentlich zugänglich dokumentiert waren. Viele der übereinstimmenden Zeilen, die Swartz fand, ließen sich so schlicht mit Anforderungen aus dem POSIX-Standard begründen und es war vollkommen legitim, ja sogar dringend erwünscht, POSIX zu folgen, selbst wenn POSIX Codezeilen forderte, die zur historischen Errungenschaft von UNIX gehören. Erschwerend kam hinzu, dass die UNIX-Philosophie möglichst einfachen, lesbaren und portierbaren Code forderte. Man kann zehn Programmierer beauftragen, ihren Code möglichst effizient zu optimieren und man wird zehn sehr verschiedene Lösungen für dieselbe Aufgabe finden. Der eine Programmierer wird sich darauf konzentrieren, die Mathematik seiner Algorithmen zu optimieren, ein anderer wird vorhersagbare Zwischenergebnisse statisch deklarieren, anstatt sie zu berechnen, ein dritter wird seinen Code in mehrere Threads verteilen, um ihn auf mehreren Prozessoren parallel auszuführen und so fort. Es gibt unzählige Möglichkeiten, Code zu optimieren. Es gibt aber nur wenige Wege, einfachen, lesbaren und portierbaren Code zu schreiben. Neben den POSIX-Anforderungen tritt hier also auch noch die UNIX-Philosophie als Gegner des Plagiatsfinders auf. Aber selbst da, wo tatsächlich eine ansatzweise belastbare Zeile identischen Codes gefunden wäre, müsste immer noch nachgewiesen werden, dass diese Zeile *ursprünglich* aus dem originalen UNIX-Code stammt, nicht aber aus dem BSD-Code, nicht aus den Modifikationen, die IBM, HP, SUN, Microsoft und viele weitere diesem Code irgendwann hinzugefügt haben, bis er dann in die UNIX-Linie zurückgeflossen ist.

Obwohl Swartz all diese Probleme bewusst waren und er sie explizit in seinem Memorandum benannte, schrieb er doch genau das, was Sabbath hören wollte: »*First many portions of Linux were clearly written with access to a copy of Unix sources. This of course would* [sic!] *be a violation of the License agreements under which Unix is distributed. Second there is some code where Linux is line for line identical to Unix*« (Swartz 1999: 3). Insgesamt konnte Swartz 48 übereinstimmende Code-Fragmente identifizieren. Alleine 26 von diesen Übereinstimmungen betrafen aber Header-Dateien, in denen üblicherweise in den öffentlich zugänglichen Standards geforderte Konstanten und mögliche Funktionscalls deklariert werden und auch die restlichen Übereinstimmungen betreffen sehr generische Basisfunktionalitäten, die sich in C kaum anders lösen ließen. Dass unter den monierten Zeilen auch etliche zu finden sind, die die Implementierung von TCP/IP-Funktionen auf Basis von Sockets betreffen, also jene erste Technologie, mit der

sich BSD entscheidend von UNIX abhob, spricht letztlich nicht für eine präzise historische Kenntnis der genealogischen Stränge auf Seiten von Swartz.

Der verdient sich dennoch sein Geld mit einem Hoffnungsschimmer, wenn er abschließend bemerkt:

> »Additionally in areas where the code is identical for compatibility reasons, the code in certain instances is character by character identical. There is a Grove Press case where the court found that making plates from the pages of an out of copyright book was a violation of law. This practice here may be similar.« (ebd.)

Alleine auf dieser vagen Hoffnung aber einen Prozess zu beginnen, wäre einem Vabanque-Spiel gleichgekommen.

»THEY ARE SMOKING CRACK«

SCO zog die Konsequenzen und suchte nun selber einen Käufer für das unglückselige UNIX. Bemerkenswerterweise zeigte sich das Unternehmen Caldera interessiert, das nicht nur neben Red Hat zu den erfolgreichen Linux-Distributoren und Supportern gehörte, sondern selber eine Ausgründung von Novell war, dem Vorbesitzer von UNIX. Sie hätten es also besser wissen müssen, kauften aber sehenden Auges 2001 die Rechte am UNIX-Code samt der Rechte am Namen SCO. Seit 2002 firmierte Caldera dann unter dem Namen *The SCO Group* und baute neben dem bis dahin erfolgreichen Linux-Geschäft einen verlustträchtigen Bereich für UNIX auf. Alle anderen historisch großen UNIX-Distributoren hatten sich längst auf den umgekehrten Weg gemacht: Novell kaufte 2003 den wichtigsten europäischen Linux-Distributor SuSE auf[35] und fuhr das UNIX-Geschäft dramatisch zurück; IBM investierte um die Jahrtausendwende eine Milliarde(!) Dollar, um die Entwicklung von Linux zu beschleunigen, das sie selber in Kundenprojekten einsetzen wollten, das aber trotz vieler Vorzüge noch nicht alle Anforderungen für große Architekturen erfüllte.[36] Einzig SUN und HP, die nicht nur Software, sondern integrierte Server aus Hardware und vorkonfigurierten Betriebssystemen vertrieben, konnten sich noch halbwegs stabil mit Unix über Wasser halten. The SCO Group unter ihrem Vorsitzenden Darl McBride aber suchte die Entscheidungsschlacht. Ohne einen wirklichen Beweis in Händen zu halten und nur auf Basis der Vagheiten in Swartz' Memorandum beklagte sich McBride über den Diebstahl an UNIX-Code durch die Linux-Gemeinde und klagte kurz darauf,

35 SuSE gehört neben Red Hat zu den ältesten Linux-Distributoren und war wegen seiner guten Deutschen Dokumentation lange Zeit die wichtigste Linux-Distribution in Deutschland.

36 IBM gibt auf der eigenen Webseite ausführlich Auskunft über die Gründe für das Linux-Engagement: http://www-03.ibm.com/linux/, 10.11.2009.

nun vor Gericht, gegen IBM.[37] IBM, so SCO, hätte nicht nur mit einer unverhältnismäßigen Investition den Markt für UNIX ruiniert, sondern auch das Eigentum von SCO aus dem IBM-eigenen und ursprünglich auf UNIX basierenden AIX in Linux einfließen lassen. Sechs lange Jahre wurde nun taktiert, prozessiert und gegenprozessiert. Alle großen IT-Unternehmen und viele der großen IT-Kunden waren involviert. Nach SCO gegen IBM folgte IBM gegen SCO, dann SCO gegen Novell und Novell gegen SCO und als SCO begann, Großkunden von Linux-Systemen zu verklagen und ihnen UNIX-Lizenzen anerpresste, damit sie ihre Linux-Systeme betreiben dürften, klagte auch noch Linux gegen SCO. Als SCO dann, um den letzten Rest an Glaubwürdigkeit zu erhalten, auch noch den eigenen Geschäftsbereich mit Linux aufgab, nahte das finanzielle Ende.[38] Immer neue Codezeilen lieferte SCO den Gerichten, niemals mit einer belastbaren Herkunft. Ein kleines Fragment, das wohl tatsächlich aus AIX stammte, kommentierte Linus Torvalds süffisant mit den Worten:

> »The code SCO showed represents an algorithm that can be used to manage a computer's memory... Not a very interesting piece of code in itself, this is very basic ›allocate a smaller chunk of memory out of a list of bigger chunks‹. The function is described in a lot of places, and exists in original Unix code and is apparently written by Ken Thompson himself. It shows up in the Lion book (a commentary on the traditional Unix), and the code is described in [Maurice J.] Bach's ›The Design of the Unix Operating System‹. In other words, it's not only 30 years old; it's actually been documented several times. It's also part of BSD Unix, which was shown to not be a derived work of the AT&T copyrights 10 years ago.«[39]

Torvalds kurgefasstes Fazit zur Schlacht von SCO gegen den Rest der Welt lautet folgerichtig: »*They are smoking crack*« und das traf wohl das, was man auch bei IBM und Novell dachte, wenn es dort auch von den Justiziaren etwas aufwändiger und kostspieliger formuliert wurde.

Wäre es darum gegangen, diesen Prozess so lange zu führen, bis mit technischen Mitteln die eine oder andere Position bewiesen worden wäre, der Prozess würde wohl nie enden wollen. Ab einer gewissen Komplexität gibt es schlicht kein technisches Verfahren, um illegitime Codeplagiate in verschiedenen Betriebssystemen zu finden, die der selben Architektur angehören und die der selben Phi-

37 Anlässlich dieser Klage ist die – inzwischen mehrfach preisgekrönte – Webseite *Groklaw* (www.groklaw.net) gegründet worden, auf der alle Schritte der SCO gegen Linux-Prozesse sorgfältig dokumentiert und kommentiert sind.

38 Dieses Ende konnte allerdings durch Investitionen der BayStar Capital noch lange heraus gezögert werden. Groklaw konnte schließlich enthüllen, dass diese Investitionen pikanterweise von Microsoft vermittelt wurden (vgl. http://www.groklaw.net/article.php?story=20061008014426 92, 10.11.2009).

39 Vgl. www.linuxtoday.com/developer/2003082101326INKNDV, 10.11.2009.

losophie folgen – *wenn Architektur und Philosophie hinreichend präzisiert sind*. Und so wurde auch dieser Prozess mit rein juristischen Mitteln beendet: Im April 2007 kann Novell nach einer jahrelangen Analyse der Verträge glaubhaft belegen, dass Sie zwar den UNIX-Code, nicht aber das Copyright an diesem Code an SCO veräußert haben. Als die Revisionsversuche durch SCO scheitern, klagt Novell die letzten Dollar aus SCOs Kriegskasse heraus, indem sie nun Lizenzgebühren für den Verkauf von UNIX-Installationen durch SCO fordern. Am 25. August 2009 entscheidet zwar das Berufungsgericht, dass die Frage der Copyright-Veräußerung erneut überprüft werden müsse, bestätigt aber Novells Anrecht auf 2,5 Millionen Euro Lizenzgebühren von SCO. Damit ist SCO endgültig zahlungsunfähig. Am 26. August wird der Insolvenzverwalter bei SCO eingesetzt, am 16. Oktober wird Darl McBride als zunächst letzter, der sich noch an den Wert des UNIX-Codes klammerte, entlassen.

EIN FAZIT

Wenn die Sache gut läuft, ist die Geschichte von UNIX hier zu Ende, Unix ist zumindest in der Reinkarnation MacOS X ein Coup gelungen[40] und die Geschichte von *nix wird durch Linux immer erfolgreicher. Vielleicht wird sich aber auch jemand aus der Konkursmasse der SCO Group bedienen und die nächste Runde einläuten. Das würde immerhin einer fast zwanzigjährigen Tradition folgen. An einem wird beides nicht vorbeiführen: *hinreichend* umfangreiche und *hinreichend* präzise formulierte Architekturstandards und Designrichtlinien nehmen jeder konkreten Instanz, in der diese Richtlinien umgesetzt werden, ihre Individualität. Das heißt nicht, dass eine Implementierung nicht in irgendeinem Sinne besser sein könnte als eine andere, wohl aber, dass der Wert der einzelnen Instanzen sich weitestgehend auf die investierte Arbeitskraft reduziert, während das schützenswerte geistige Eigentum in den Standards verbleibt. Plagiate auf Ebene solcher Quellcodes suchen zu wollen, ist zum einen sinnlos und zum anderen hoffnungslos, weil jede potenziell kopierte Codezeile nur dann werthaltig sein kann, wenn dieser Wert schon in der Architektur vorgegeben ist, die der Code reproduziert. Dem entgegenzuwirken ist nur dort möglich, wo die definierten Architekturen ignoriert und die Standardisierung wieder im Code selber stattfinden würde – wo der Code die Architektur ist. Solcher Code wäre hinreichend individuell, um mit Swartz' *ef* oder vergleichbaren technischen Mitteln unzulässige Kopien zu identifizieren. Ob diese Möglichkeit des Schutzes die notwendige Aufgabe der Möglichkeit zur Kollaboration und Kooperation oder, im Web 2.0-Slang, zur *Community*-

40 Bemerkenswert im Zusammenhang dieses Artikels ist, dass Apple für die proprietären Teile von MacOS X, insbesondere für die Cocoa-API, mit Objectice-C eine extrem selten verwendete Programmiersprache verwendet. Cocoa-Plagiate dürften also im Unterschied zu den OpenSource-Bestandteilen sehr schnell aufgespürt und zweifelsfrei identifiziert werden können.

Bildung rechtfertigt, wird jedes Unternehmen selbst entscheiden müssen, gleich, ob es Software produziert oder irgendein anderes Gut.

LITERATURVERZEICHNIS

Brooks, Frederick P. (1995): *The Mythical Man-Month: Essays on Software Engineering, 20th Anniversary Edition,* Reading, MA: Addison-Wesley.

Ceruzzi, Paul E. (2003): *A History of Modern Computing*, Cambridge, MA: MIT Press.

Clukey, Lee Paul (1985): *UNIX & XENIX demystified*, Blue Ridge Summit, PA.: TAB Books.

Flamm, Kenneth (1988): *Creating the Computer: Government, Industry, and High Technology*, Washington, DC: Brookings Institution.

Kernighan, Brian/Ritchie, Dennis (1988): *The C Programming Language*, 2nd Edition, Englewood Cliffs, NJ: Prentice Hall.

Kittler, Friedrich (1993): »Es gibt keine Software«, in: ders.: *Draculas Vermächtnis. Technische Schriften*, Leipzig: Reclam, S. 225-242.

Lanzerotti, Mary Y. (Hg.) (2006): *The Technical Impact of Moore's Law. IEEE solid-state circuits society newsletter*, Vol. 20, No. 3.

Lehtinen, Rick/Russell, Deborah/Gangemi, G. T. (2006): *Computer Security Basics*, 2nd Edition, Beijing u.a.: O'Reilly.

Ng, Kia/Nesi, Paolo (2008): *Interactive Multimedia Music Technologies*, Hershey, PA: Information Science Reference.

Raymond, Eric S. (2004): *The Art of Unix Programming*, Boston u.a.: Addison-Wesley.

Schäfer, Fabian (2007): *Der virale Effekt. Entwicklungsrisiken im Umfeld von Open Source Software. Schriften des Zentrums für angewandte Rechtswissenschaft Karlsruhe*, Karlsruhe: Universitätsverlag.

Schwabach, Aaron (2006): *Internet and the Law. Technology, Society, and Compromises*, Santa Barbara u.a.: ABC-CLIO.

Shore, Jim/Warden, Shane (2007): *The Art of Agile Development*, Beijing u.a.: O'Reilly.

Stare, Metka/Rubalcaba, Luis B. (2008): »Research on Services: From Exploring the ›Residual‹ to Services Science«, in: Stauss, Bernd/Engelmann, Kai/Kremer, Anja et al. (Hg.): *Services Science. Fundamentals, Challenges and Future Developments*, Berlin u.a.: Springer, S. 41-54.

Swartz, Robert (1999): »Memorandum«, http://www.sco.com/company/legal/update/memorandum_19991004.pdf, 10.11.2009.

Stevens, W. Richard (1995): *TCP/IP Illustrated, Volume 2: The Implementation*, Reading, MA: Addison-Wesley.

Torvalds, Linus/Diamond, David (2002): *Just for Fun. Wie ein Freak die Computerwelt revolutionierte*, München u.a.: Hanser.

WEB-RESSOURCEN

American National Standards Institute (ANSI): http://www.ansi.org/

Austin Group der Open Group: http://www.opengroup.org/austin/

Gesetz über Urheberrecht und verwandte Schutzrechte (UrhG): http://www.gesetze-im-internet.de/bundesrecht/urhg/gesamt.pdf

GNU General Public License, Version 2 (GPL v2): http://www.gnu.org/licenses/old-licenses/gpl-2.0.txt

GNU General Public License, Version 3 (GPL v3): http://www.gnu.org/licenses/gpl.txt

Groklaw: www.groklaw.net

Hurd-Kernel: http://www.gnu.org/software/hurd/

IBM und Linux: http://www-03.ibm.com/linux/

UNIX-Seiten der Open Group: http://www.unix.org

Wikipedia: http://en.wikipedia.org/wiki/ {AT%26T|Bell_Labs|Posix|Santa_Cruz_Operation|Unix|Xenix}