

1 Open Source im Unternehmen

abgedruckt in: „Linux im Unternehmen“, hrsg. von A. v. Raison und R. Schönfeldt, dpunkt Verlag, Heidelberg 2001.

Zusammenfassung:

Open Source-Software, also im Quelltext offengelegte und frei erhältliche Software, bietet besondere Chancen sowohl für deren Produzenten als auch für die Konsumenten, die dieser Beitrag beleuchten will. Hersteller müssen sich bei der Umstellung auf Open Source auf einen Kulturwandel gefasst machen, mit dem andere Geschäftsmodelle und veränderte Rechtsbeziehungen verbunden sind. Anwender bekommen zwar die Programme kostenlos, haben dafür aber unter Umständen andere Folgekosten zu tragen und mit völlig neuen Situationen im Gewährleistungsfall zu rechnen. Da das Internet bereits für den Durchbruch von Open Source gesorgt hat, werden die Unternehmen gewinnen, die sich frühzeitig auf die veränderten Umstände einstellen.

Heute vergeht kaum eine Besprechung zwischen Managern im IT-Bereich, bei der nicht das Thema „Open Source“ zur Sprache käme. Software, die frei verfügbar und deren Quellcode offengelegt ist, übt eine starke Anziehungskraft aus. Manche sehen darin die Lösung all ihrer Probleme, die meisten Verantwortlichen stehen der Entwicklung jedoch mit gemischten Gefühlen gegenüber, da sie spüren, dass Open Source ihr gesamtes Weltbild verändern wird. Oftmals entspringt dieses Unbehagen auch einer gewissen Unkenntnis der Materie, denn bislang haben sich Open Source-Entwicklungen stets auf einer Ebene abgespielt, die kaum geschäftsrelevant war und daher das Management nicht sonderlich interessierte. Als erstes wollen wir daher die Frage klären, was unter „Open Source-Software“ genau zu verstehen ist.

1.1 Was ist Open Source-Software?

Open Source ist im Grunde nichts neues. Als ich begonnen habe, mich für Computer zu interessieren, gab es für meinen Sinclair Spektrum und Commodore 64 sowie für andere Computer eine Menge Zeitschriften, die Monat für Monat Programme für diese in Form von Quellcode abdruckten. Wir Anwender mussten die Programme dann eintippen, hatten aber den gesamten Code zur Verfügung und konnten daran nach Belieben Anpassungen und Erweiterungen vornehmen. Es gab auch damals schon die Möglichkeit, mittels Datenfernübertragung und verschiedener elektronischer schwarzer Bretter Code in digitaler Form auszutauschen, was in Deutschland aber aufgrund des Post-Monopols, das zu einer schlechten Verfügbarkeit von Modems und hohen Kosten führte, kaum Bedeutung erlangte. Die Idee als solche blieb aber bestehen.

1.1.1 GNU und GPL-Software

Etwa gleichzeitig, im Jahr 1984, rief Richard Stallman am MIT das Projekt „GNU“ (ein rekursives Akronym aus „GNU's not Unix“) ins Leben, das das Ziel hatte, ein Unix-artiges Betriebssystem zu schaffen – ein Ziel übrigens, das durch den Linux-Kernel tatsächlich verwirklicht wurde, weshalb heute Linux korrekterweise als „GNU/Linux“ zu bezeichnen ist. Die Ideale von GNU wurden zu den Leitlinien der gesamten heutigen Open-Source-Bewegung: Software sollte „frei“ sein, wobei „frei“ im Sinne von „freier Rede“ und nicht von „Freibier“ zu verstehen ist. Bei freier Software hat der Benutzer das Recht, sie zu kopieren, zu verbreiten, zu analysieren und zu verbessern. Die unbedingte Voraussetzung dafür ist, dass der Anwender Zugang zum Quellcode der Software erhält.



Abbildung 1: Das Gnu ist das Logo des GNU-Projekts

Um die Ideale zu bewahren und zu fördern, gründete Stallman zusammen mit einigen anderen die Free Software Foundation (FSF, www.fsf.org). Die FSF arbeitete unter anderem eine Lizenzregelung aus, mit der sich die Freiheit an der Software erhalten lassen und gleichzeitig das Urheberrecht gesichert bleiben sollte. Diese ist heute in ihrer Version 2 von 1991 sehr weit verbreitet und unter der Bezeichnung „GNU General Public License“ (GPL) bekannt [1]. Sie erlaubt es dem Anwender, Software, die darunter lizenziert ist, zu modifizieren, zu kopieren und zu verbreiten – vorausgesetzt, dass die dabei entstehende ‚abgeleitete Software‘ ebenfalls wieder unter der GPL veröffentlicht wird. Damit ist immer auch eine Freigabe des Quellcodes verbunden. Sie erhalten also kostenlos GPL-lizenzierten Code als Basis für ein eigenes Programm, müssen als Gegenleistung jedoch die eigenen Ergänzungen auch wieder jedermann zur Verfügung stellen und können damit andere nicht daran hindern, den gleichen Nutzen aus Ihrem Code zu ziehen, wie Sie das bei der vorherigen Version getan haben. Diese Vorschrift bezieht sich allerdings nur auf den Umgang mit dem Code in der Öffentlichkeit; was Sie damit intern, also beispielsweise innerhalb Ihrer Firma machen, steht Ihnen völlig frei. Sie müssen dazu auch niemanden um Erlaubnis fragen. Die Lizenzbestimmungen greifen erst dann, wenn Sie die Software wieder verbreiten, etwa als Produkt oder Teil eines Produktes.

Die GNU General Public License

Bekanntestes Beispiel für GPL-lizenzierte Software ist der Linux-Kernel. Aber auch die gesamte GNU-Software vom Compiler GCC über GNU-make bis zu GNU-Chess, die Benutzeroberflächen KDE und Gnome sowie neuerdings „StarOffice“ (von Sun Microsystems, ehemals von StarDivision) sind unter den Bedingungen der GPL veröffentlicht.

Beispiele für GPL-Software

Steht eine Bibliothek unter der GPL, so fallen alle Programme, die diese Bibliothek dazulinken – gleichgültig ob statisch oder dynamisch –, ebenfalls unter die Veröffentlichungspflicht, da sie als „abgeleitetes Werk“ anzusehen sind. Aus diesem Grund hat man für Bibliotheken eine Abschwächung der GPL formuliert, die „Library GPL“ (LGPL). So lizenzierte Software (im allgemeinen sind dies Bibliotheken) darf ohne Weiteres zu proprietären Anwendungen dazugelinkt werden, ohne dass diese wiederum automatisch unter die GPL fallen müssten. Bekanntes Beispiel ist die GNU C-Bibliothek Glibc, die die Programmierschnittstelle zwischen dem Linux-Kernel und Anwendungsprogrammen bereitstellt.

Die „Library GPL“

Ein Problem, das viele Leute mit der GPL haben, ist, dass die Grenzen, was unter „abgeleiteter Software“ zu verstehen ist, nicht immer leicht zu ziehen sind. Linus Torvalds hat zu den Lizenzbestimmungen des Linux-Kernels daher noch den Satz hinzugefügt: „Dieses Copyright umfasst *keine* Anwenderprogramme, die die Kerneldienste durch normale Systemaufrufe verwenden – dies wird lediglich als normale Benutzung des Kernels angesehen und fällt daher *nicht* unter den Begriff der ‚abgeleite-

Klärung der GPL für den Linux-Kernel

ten Software'“. Daher sind Anwenderprogramme, die nur die normalen Kernel-Systemaufrufe verwenden (was ohnehin bereits über die übliche Programmierung über die Glibc hinausgeht), auch als geschlossene Software ohne lizenzrechtliche Probleme realisierbar.

1.1.2 Andere Open Source-Ansätze

Die GPL greift sehr weit in den Entstehungs- und Verbreitungsprozess von Software ein. Andere Gruppen und Institutionen, die ebenfalls die positiven Auswirkungen von offenen Quellen erkannten, wollten nicht immer so weit gehen und entwickelten daher eine Reihe anderer Lizenzmodelle ([2] gibt einen Überblick). Allen gemeinsam ist die Freigabe des Quellcodes der Software, so dass jeder Interessierte sich von der Wirkungsweise der Programme überzeugen kann. Die Unterschiede liegen darin, was mit Änderungen passieren soll und zu welchem Zweck die Software genutzt werden darf. Beispielsweise bestimmt die „Qt Public License“ QPL, unter der lange Zeit die Qt-Klassenbibliothek des norwegischen Softwarehauses Trolltech stand, dass Änderungen nur als Patches und nicht als Gesamtpaket verbreitet werden dürfen und dass die Software nur so lange frei genutzt werden kann, solange das darauf aufbauende Produkt ebenfalls frei ist.

Open Source-Politik von Softwarehäusern

Da der Erstentwickler zunächst mal alle Rechte an seiner Software hat, steht es ihm frei, welche Bedingungen er an die Freigabe seines Codes knüpft. Einige Firmen wie beispielsweise Apple, Sun oder Netscape wollten auf der Open Source-Welle mitreiten, außer dem Quellcode aber kaum andere Rechte an ihrer Software aufgeben. Erst als die Open Source-Gemeinde ihre Kritik an der Vorgehensweise der Firmen immer lauter verbreitete, lenkten die Unternehmen ein und änderten ihre Bedingungen.

Prinzipiell zweifeln einige Juristen an, dass die GPL einer gerichtlichen Prüfung standhalten würde. Für die Praxis ist dies jedoch unerheblich. Die Möglichkeiten zur Beeinflussung der öffentlichen Meinung über das Internet sind heute so vielfältig, dass Firmen, die offen gegen die GPL verstoßen, schnell mit mehr negativen Schlagzeilen konfrontiert werden können, als ihnen lieb sein kann.

Beispiele für Open Source-Projekte

Einige Beispiele für Open Source-Projekte, die unter anderen Bestimmungen als der GPL veröffentlicht werden, sind ZOPE von Digital Creations, PHP von Zend Technologies, Enhydra von Lutris Technologies oder Mozilla von Netscape/AOL.

1.2 Vorteile von Open Source

Open Source-Entwicklung bietet im Gegensatz zu geschlossenem Code eine Reihe von nicht zu unterschätzenden Vorteilen, die wir im folgenden näher untersuchen wollen.

1.2.1 Höhere Qualität

Eine der renommiertesten Vorreiter der Open Source-Bewegung ist Eric Raymond, der mit seinen Essays „The Cathedral and the Bazaar“ [3] und „The Magic Cauldron“ [4] viele wertvolle Argumente zur Diskussion beigetragen hat.

Er sieht einen der wichtigsten Vorteile von Open Source im „Peer review“-Prinzip. Das bedeutet, dass Programme anhand ihres Code von anderen Experten beurteilt werden können. Durch diese kritische Überprüfung können alle Arten von Fehlern relativ rasch aufgedeckt und beseitigt werden. Denn die Wahrscheinlichkeit, dass ein Fehler unbemerkt bleibt, ist bei mehreren Hundert Programmierern um Größenordnungen geringer als bei ein paar wenigen – sofern ein Code-Review bei der geschlossenen Entwicklung überhaupt stattfindet. Aus diesem Grund ist das Open Source-Modell besonders für Infrastruktur-Komponenten wie Betriebssystem-Kernels, Treiber oder Netzwerkdienste sinnvoll.

Das „Peer review“-Prinzip

1.2.2 Höhere Sicherheit

Nicht nur für die Qualität ist das „Peer review“ vorteilhaft, auch die Sicherheit profitiert davon. Denn eine wirklich ernsthafte Sicherheitsüberprüfung muss sich auch auf den Code beziehen; nur Algorithmen und Implementierungen, die von verschiedenen Seiten als sicher eingestuft werden, kann letztendlich vertraut werden. Mit diesen Argumenten warnen Organisationen wie das deutsche Bundesamt für Sicherheit in der Informationstechnik (BSI), die Europäische Kommission oder der amerikanische Geheimdienst NSA vor Betriebssystemen wie Microsoft Windows 2000, deren Code nicht öffentlich ist.

Sicherheitsüberprüfungen

1.2.3 Wiederverwendbarkeit für größere Komplexität

Wenn der Code einer Software offen liegt, kann jeder sich die darin verfolgte Problemlösung ansehen und daraus lernen. Eine bereits vorhandene und bewährte Lösung kann dann immer wieder verwendet werden. Der

Einzelne muss sich nicht jedes Werkzeug selbst herstellen, sondern kann dieses einsetzen, um darauf eine komplexere Lösung zu erstellen. Dieses allgemeine Innovationsprinzip, das die technische Entwicklung der Menschheit charakterisierte, ist mit geschlossener Software kaum möglich.

1.2.4 Höhere Reife durch Unabhängigkeit

Markteinführung oft nicht synchron mit Entwicklung

Bei einem Software-Produkt, für das Lizenzgebühren verlangt werden, ist oft der Zeitpunkt der Markteinführung ausschlaggebend für den Erfolg. Leider ist der Entwicklungszyklus oft nicht synchron damit, das heißt, das Produkt ist bei der Markteinführung eigentlich noch gar nicht reif dafür. Das Ergebnis ist die bekannte „Bananen-Software“, die grün ausgeliefert wird und erst beim Anwender reifen soll. Der ökonomische Schaden, der durch die massenhaften Probleme der Benutzer entsteht, ist kaum zu beziffern.

Reife und Qualität bei OS-Releases höher

Da die Open Source-Entwickler ihre Arbeit nur aus Spaß am Programmieren oder aus Prestige-Gründen verrichten, sehen sie in der Reife und Qualität ihres Produktes einen sehr wichtigen Aspekt. Und da sie keinem Produktmanager und keiner Geschäftsleitung verantwortlich sind, entscheiden sie über die Freigabe einer Version im allgemeinen nach rein technischen Gesichtspunkten. Ein Beispiel dafür ist der Linux-Kernel, insbesondere die Version 2.4. Seine Freigabe kam über ein Jahr später als ursprünglich angekündigt, da die Entwickler ihn so lange zurückhielten, bis sie von seiner Stabilität überzeugt waren.

Beispiel Mozilla

Bei Mozilla konnte man die Probleme dieser Argumentation mit einer kommerziellen Betreuer-Firma erleben. Obwohl die Entwickler von einer Freigabe einer neuen Version des Internet-Browsers abrieten, setzte sich die Muttergesellschaft AOL darüber hinweg und brachte „Netscape 6“ auf den Markt. Wie erwartet war das Produkt reichlich instabil und an vielen Stellen schlicht „unfertig“. Das Vertrauen zum Netscape-Browser könnte damit nachhaltigen Schaden genommen haben.

1.3 Open Source aus Sicht des Herstellers

In vielen Unternehmen wird Software entwickelt, auch in solchen, die nicht unmittelbar damit ihr Geld verdienen. Blickt man in die Stellenangebote für Programmierer, so werden immer noch die meisten Leute dafür gesucht, Software zur ausschließlichen Nutzung im Haus zu entwickeln. Ein kleiner Teil der Firmen erwirtschaftet jedoch nach wie vor ihren Er-

trag durch den Verkauf von Software-Lizenzen. Aufgrund des Erfolges und der unbestreitbaren Vorteile des Open Source-Ansatzes stellen sich heute immer mehr Produkt-Verantwortliche die Frage, ob sie ihre Software freigeben sollen. Ich will Ihnen hier ein paar Denkanstöße geben und Ängste nehmen.

1.3.1 Geschäft mit Lizenzen

Zunächst mal geht die Gegenüberstellung von freier und kommerzieller Software oft von der Irrmeinung aus, dass Software ein industrielles Produkt wie jedes andere auch sei. Bei einem solchen Produkt, etwa einem Telefon oder einem Speicherchip, ist die hergestellte Stückzahl annähernd direkt proportional zu den Kosten, da jeweils Rohstoff- und Fertigungskosten anfallen. Bei Software ist dies nicht der Fall, da ihre Vervielfältigungskosten vernachlässigbar sind. Die Käufer sind nur bereit, solange für ein Produkt zu bezahlen, wie sie sich eine eigene Zeit- oder Aufwandsersparnis beziehungsweise einen künftigen „Service“ davon versprechen. Software eines Herstellers, den es nicht mehr gibt, ist so gut wie nichts mehr wert.

Software als industrielles Produkt?

Brian Behlendorf, einer der Mitbegründer des Web-Servers Apache, beschreibt in dem höchst lesenswerten Artikel „Open Source as a Business Strategy“ [5] einen Datenbankhersteller, der etwa 40 % seines Umsatzes mit dem Verkauf von Lizenzen erwirtschaftet, den Rest mit Support, Consulting, Entwicklungswerkzeugen und Bibliotheken. Die Rechnung ist nun ganz einfach: Wenn die Datenbank als Open Source freigegeben wird, müssen nur doppelt so viele Kunden wie momentan gefunden werden, die für Consulting und Support bezahlen, um den Umsatz trotz des Wegfalls der Lizenzeinnahmen um 20% steigern zu können. Wenn die Datenbank attraktiv ist und zudem noch frei, werden sich vermutlich sogar sehr viel mehr Kunden dafür interessieren.

Geschäft ohne Lizenzeinnahmen

Dass dieses Modell nicht nur hypothetisch ist, sondern auch in der Praxis funktioniert, zeigt das Beispiel der CAD-Lösung „CAS.CADE“ der EADS Matra Datavision [6]. Die Software, die 1999 pro Kernel noch fünfstelligen Lizenzeinnahmen brachte, wurde Ende 99 als Open Source freigegeben. Gleichzeitig wurde über entsprechende Web-Sites der Aufbau einer Community begonnen, auch in Zusammenarbeit mit den bisherigen Kunden. Bereits im darauffolgenden Jahr konnte Matra seinen Umsatz trotz der entfallenen Einnahmen steigern. Obwohl die hauseigenen Entwicklerkapazitäten reduziert wurden, bietet das nun Open CASCADE genannte Produkt mehr Funktionalität als die Konkurrenz. Die Kunden nutzen das Support-, Consulting- und Trainingsangebot rege, über 200 Downloads pro Woche belegen steigendes Interesse.

Beispiel Open CASCADE

1.3.2 Sicherung des geistigen Eigentums

Auch freie Software hat Urheber. Durch die Veröffentlichung als Open Source geben diese ihre Urheberrechte daran in keiner Weise auf, ebenso wenig wie beispielsweise ein Buch-Autor. Für viele Software-Verantwortliche ist gerade dieser Aspekt besonders heikel. Dabei muss man allerdings meist genau hinschauen, ob die Befürchtungen wirklich zu Ende gedacht sind oder nicht.

Closed Source als Wettbewerbsvorteil

Der häufigste Grund, warum man die Quelle einer Software nicht veröffentlichen möchte, ist, dass man darin einen Wettbewerbsvorteil gegenüber seinen Konkurrenten sieht, der man nicht leichtfertig aufs Spiel setzen möchte. Das kann im Einzelfall tatsächlich ein Argument sein, das den Ausschlag für eine Entscheidung gegen eine Open Source-Politik gibt. Wie wir am Beispiel von Open CASCADE gesehen haben, ist das allerdings nur in den wenigsten Fällen in letzter Konsequenz richtig. Vielleicht lassen sich durch eine Veröffentlichung als Open Source sogar interne Entwicklungskosten einsparen und gleichzeitig die Funktionalität vergrößern. Außerdem wird die Verbreitung einer guten freien Software immer um ein Vielfaches größer sein als die einer proprietären.

Trend: Software als Service

Die gesamte IT-Industrie scheint sich derzeit darauf zuzubewegen, Software nicht mehr als Produkt, sondern als Dienstleistung zu betrachten. Auch für den Kunden sind ja die Lizenzausgaben nur ein kleiner Faktor der Gesamtkosten einer Software. Es wird künftig verstärkt Aufgabe des Herstellers sein, für sich selbst einen größeren Anteil an den Lebenszeitkosten einer Software zu sichern, als das bisher der Fall war. Damit sinkt natürlich auch die Bedeutung der Lizenzeinnahmen.

1.3.3 Motivation für Open Source

Welche Gründe sprechen nun aus Sicht eines Software-Herstellers dafür, ein Programm als Open Source zu veröffentlichen und damit alle Welt in seine Karten, sprich: Quellen, sehen zu lassen?

Marketing und PR

Früher entstanden die meisten Open Source-Projekte dadurch, dass ein Programmierer eine Idee verwirklichen wollte oder ein bestimmtes Tool benötigte, das er sich selber schreiben musste, dann aber mit anderen teilen wollte. Wenn eine Firma heute ein solches Projekt ins Leben ruft, geschieht dies kaum noch aus technischen, dafür meist aus Marketing-Überlegungen. Auf diese Weise kann sich das Unternehmen nicht nur als offen und innovativ hervorheben, sondern auch bestimmte Benutzergruppen an sich binden oder sich in neuen Märkten positionieren.

Setzen von Standards

Außerdem können durch Open Source-Programme eigene Standards mit viel größerer Breitenwirkung und in kürzeren Zeiträumen auf dem

Markt etabliert werden, als dies mit kostenpflichtiger Software möglich wäre. Diese Strategie kann man beispielsweise bei Sun mit Java und bei IBM mit Internet- und XML-Software beobachten. Letztlich versichern zwar alle Hersteller, keine proprietären Standards zu wollen, sondern sich an offene Standards der internationalen Gremien (IETF, W3C etc.) zu halten. Da die Verabschiedung von Standards durch diese indessen meist recht lange dauert, kann der Hersteller seine eigenen Vorstellungen am Markt und bei der Standardisierung am besten durchsetzen, der über die größte Anwenderbasis verfügt.

Damit ist nämlich auch gleich das Feld für die nächste Stufe bereitet: Der Hersteller findet bei genügender Akzeptanz des von ihm favorisierten Standards einen lohnenden Markt für kommerzielle Produkte vor, die darauf aufbauen und die Möglichkeiten der Open Source-Programme ergänzen beziehungsweise erweitern. Das können größere kostspielige Software-Pakete, etwa für die Entwicklung oder Administration, genauso sein wie Hardware. Auch dafür ist IBM mit der WebSphere-Produktreihe, die auf Java- und Internetstandards setzt, oder der Linux-Portierung auf AS/400 ein gutes Beispiel.

Geschäft mit darauf aufbauenden Produkten

Für die Anwender ist es mit der kostenlosen Software allein oft nicht getan. Je komplexer die Anwendung, desto mehr Bedarf besteht an Unterstützung bei der Anpassung an die Umgebung des Benutzers und an der Erarbeitung maßgeschneiderter Lösungen. Obwohl dies die klassische Einnahmequelle einer Open Source-Firma ist, ist diese Strategie nach wie vor lohnenswert.

Geschäft mit Services

Die Entwicklungsleistung kann dabei auch teilweise von einzelnen Anwendern kommen. Für manche Kunden kann das Programm so wichtig sein, dass sie eigene Ressourcen für dessen Pflege und Ausbau bereitstellen – Erweiterungen, die sie aufgrund des Lizenzmodells meist wieder an den Hersteller zurückmelden müssen. Auf diese Weise können also Entwicklungskosten vom ursprünglichen Produzenten auf seine Kunden verlagert werden – in vielen Fällen ein nicht unwesentliches Argument.

Anwender erweitern Features

Aber nicht nur die Entwicklung kostet Geld; oft ist der laufende Unterhalt für die Pflege einer Software sogar noch teuer. Viele Firmen pflegen Software-Produkte nicht, weil sie mit dem laufenden Lizenzverkauf noch etwas verdienen, sondern nur, um bestehende Kunden zu bedienen. In solchen Fällen bietet sich die Freigabe der Software als Open Source an, da man damit zumindest teilweise die Pflegeaufwände auf andere verteilen kann. Dazu ist allerdings auch der vorbereitende Aufbau einer entsprechenden Community nötig [7]. Ein prominentes Beispiel für diesen Schritt ist AOL/Netscape mit der Freigabe des Mozilla-Browsers – auch wenn dabei in der Folge noch ein paar andere Einflüsse hinzukamen.

Hersteller entbindet sich von der laufenden Pflege

Schließlich ist noch zu bedenken, dass auch in einem Unternehmen hinter einer Software nicht immer riesige Teams stehen, sondern oft nur

Unabhängigkeit von einzelnen Entwicklern

zwei oder drei Mitarbeiter. Und von diesen hängt die Software dann auch ab. Selbst wenn die Programme ausreichend dokumentiert sind, lassen sie sich im allgemeinen nicht ohne Weiteres durch andere übernehmen. Falls also einer der Mitarbeiter (oder gar alle) die Firma verlassen oder eine neue Aufgabe übernehmen, ist die Gefahr groß, dass die Anwendung nicht gepflegt wird, hinter den sonstigen technischen Entwicklungen hinterherhinkt und langsam unbrauchbar wird. Durch die Veröffentlichung der Software als Open Source kann man dieses Problem vermeiden, denn man übergibt das Programm einer – hoffentlich – größeren Gemeinschaft, die sich um den Fortbestand (und somit um die Sicherung der ursprünglichen Investition in die Entwicklung) kümmern wird. So werden vielleicht nicht unmittelbar Kosten eingespart, dafür aber das Risiko der Unbrauchbarkeit auf eine breite Anwenderbasis außerhalb der eigenen Firma verteilt und die Abhängigkeit von einzelnen Entwicklern reduziert.

1.3.4 Open Source bei Embedded Systemen

Software ist nicht immer das, was auf einem PC abläuft und auf einem Monitor angezeigt wird. Wir sind überall von so genannten „embedded Systemen“ umgeben (siehe auch [8] für einen Überblick), von der Waschmaschine über das Handy bis zum Auto, wo die Software in ein Gerät eingebettet ist und für den Benutzer des Geräts nur sehr vage sichtbar ist. Aufgrund seiner Stabilität, Flexibilität und Offenheit erlangt Linux als Betriebssystem für solche Embedded Systeme immer größere Beliebtheit. Hierbei stellt sich die Frage nach der GPL-Konformität und der Zulässigkeit von proprietärer Software in verstärktem Maße, da die Anwendungen auf solchen Geräten im allgemeinen sehr viel näher an der Hardware arbeiten und daher auf tieferer Ebene mit dem Betriebssystem kommunizieren müssen, als das bei Server- oder Desktop-Anwendungen der Fall ist.

Kernel-Module

Aus obiger Diskussion um die GPL ist klar, dass Anwendungen, die auf der Glibc aufsetzen, problemlos als Closed Source realisierbar sind. Ebenso klar ist, dass direkte Modifikationen am Linux-Kernel aufgrund seiner GPL-Lizenz veröffentlicht werden müssen. Interessanter sind da die Kernel-Module und Treiber, also Software, die nicht unmittelbar Teil des Kernels ist, jedoch sehr eng mit diesem zusammenarbeitet. Gemäß der oben zitierten Klärung der GPL für den Kernel gilt, dass Module und Treiber, die die normalen Kernel-Schnittstellen verwenden, auch als nicht-öffentliche Software möglich sind. Allerdings stehen viele Kernel-Entwickler dieser Ausnahme nach wie vor skeptisch gegenüber, so dass hier mit Bedacht vorgegangen werden sollte. Bei einem Modul, das nicht

die Standard-Schnittstellen verwendet, ist die Situation noch heikler; hier sollte stets auf eine Open Source-Politik geachtet werden.

Generell gilt ohnehin, dass ein Betriebssystem eine gemeinsame Basis für alle Anwendungen zur Verfügung stellen sollte – nicht nur für eine bestimmte. Trennen Sie also Ihre Software durch eine intelligente Architektur möglichst in Hardwarefunktionen auf unterer Ebene (wo höchst selten geschäftssensibler Code enthalten ist) und Anwendungsprogramme auf höherer Ebene, die Sie dann auch unbedenklich als proprietäre Software herausgeben können.

Architekturelle Trennung

Wenn Sie dann noch Ihre Treiber als Open Source veröffentlichen, haben Sie auch deren Weiterbestand gesichert, denn so können diese leicht von anderen angepasst werden, wenn sich die entsprechenden Kernel-Schnittstellen in einer der nächsten Versionen ändern. Linux-Treiber, die nur in binärer Form vorliegen, sind aufgrund der Dynamik der Kernel-Entwicklung – gerade im Embedded-Bereich – reichlich hinderlich.

Treiber stets als Open Source

1.3.5 Neue Geschäftsmodelle

Wie Sie vielleicht schon erkannt haben, führt Open Source zu anderen Geschäftsmodellen als der reine Lizenzverkauf. Diese wurden von vielen Publikationen zur Genüge beschrieben (etwa in [3], [4], [10], [11] und [12]), so dass ich hier nur die wichtigsten wiedergeben will:

- ❑ "*Support Sellers*", wobei die Einnahmen vom Medienvertrieb (wie bei RedHat) oder von Training und Consulting herrühren (wie bei VA Linux)
- ❑ "*Loss Leader*", wo ein kostenloses Open Source-Produkt den Markt für eine kommerzielle Software bereiten soll (wie bei IBMs XML-Tools)
- ❑ "*Widget Frosting*", wo Hardware-Hersteller die zusätzliche Software wie Treiber, Konfigurationstools etc. als Open Source freigeben, da sie ja ihr Geschäft mit Hardware machen. Ein jüngstes Beispiel sind die Deskjet-Treiber für HP-Drucker.
- ❑ "*Accessorizing*", wobei die Firma Umsatz mit Büchern, Konferenzen oder speziell angepassten Hardware-Angeboten macht, z.B. O'Reilly oder Dell.
- ❑ "*Service Enabler*"; hier ist der Zugang zu Online-Diensten eine Open Source-Software, der Inhalt allerdings kostenpflichtig. Dieses Modell wäre für AOL sinnvoll.

- ❑ "*Brand Licensing*", wo eine Firma zwar die Software bzw. die Software-Technologie als Open Source freigibt, damit aber gleichzeitig ein Markenzeichen etabliert, für dessen Zertifizierung sie Geld verlangt, so wie es etwa Sun Microsystems mit Java machen könnte.
- ❑ "*Sell It, Free It*" ist ein häufiges Modell, wo eine Software zunächst ein kommerzielles Produkt ist, dann aber als Open Source erscheint, wenn es in die Geschäftsplanung des Herstellers passt. Interbase und StarOffice sind Beispiele dafür.

1.3.6 Grenzen und Gefahren des Open Source-Modells

Kriterien für Open Source-Kandidaten

In [4] fasst Eric Raymond die wesentlichen Kriterien, wann eine Software ein Kandidat für eine Freigabe als Open Source ist, wie folgt zusammen:

- ❑ Zuverlässigkeit, Stabilität oder Skalierbarkeit sind kritisch.
- ❑ Die Korrektheit des Designs oder der Implementierung können nicht zuverlässig ohne „peer review“ überprüft werden.
- ❑ Die Software ist kritisch für die Kontrolle des Anwenders über sein Geschäft.
- ❑ Die Software baut eine gemeinsame Rechen- oder Kommunikationsinfrastruktur auf.
- ❑ Die Schlüsselfunktionen sind Teil des allgemeinen Ingenieurswissens.

Wenn Sie sich daran orientieren und zudem noch beurteilen, wie viel Geschäft Sie tatsächlich noch mit einigen Programmen machen, werden Sie sicher eine ganze Reihe von Software-Paketen bei sich finden, die sich als Open Source eignen würden.

Motivation der freiwilligen Mitarbeiter

Das heißt auf der anderen Seite allerdings nicht, dass Open Source in Allheilmittel für alle Art von Software ist. Man kann nicht seinen Code auf einen Server legen und erwarten, dass am nächsten Morgen bereits viele tolle neue Features eingebaut sind. Damit eine Software ein erfolgreiches Open Source-Projekt wird, müssen Sie auf die Motivation der anvisierten freiwilligen Mitarbeiter achten. Da diese kein Geld erhalten, wollen sie einen anderen Nutzen daraus ziehen. Zwei Gründe sind dafür hauptsächlich ausschlaggebend:

- ❑ *Prestige*: Wenn Programmierer sich in ihrer Freizeit mit Open Source-Projekten beschäftigen, so tun sie das oftmals, um sich einen Namen zu machen. Das setzt voraus, dass es sich bei dem konkreten Projekt um einen attraktiven, herausfordernden Inhalt handelt. Wenn abzusehen ist, dass für das Produkt nur wenig Inte-

ressenten vorhanden sein werden, ist es auch schwer, dafür Programmierer zu motivieren.

- **Anwendernutzen:** Bei Anwendungen im Geschäftsumfeld spielt es oft noch eine Rolle, ob sich eine Mitarbeit für das Anwenderunternehmen lohnt, also beispielsweise eine Anpassung der Applikation an ihre besondere Umgebung einen höheren Nutzen bringen würde. Nur dann lassen sich die Kunden in die Programmierarbeit integrieren.

Wichtig ist es also, für das Projekt frühzeitig eine Community von Interessierten aufzubauen [7]. Das ist bei recht kleinen Projekten unter Umständen schwierig, so dass dafür Open Source nicht immer der richtige Weg ist. Denn wenn Open Source nur bedeutet, dass sich jeder bedienen kann und niemand etwas in Form von Mitarbeit zurückgibt, findet schnell ein Ausverkauf der Interessen des Initiators statt.

Aufbau einer Community

Wichtig ist stets, das Projekt zunächst möglichst bekannt zu machen und eine Infrastruktur für die Mitarbeit von Externen aufzubauen. Ein bekanntes Forum für Open Source ist das von VA Linux betriebene SourceForge [8]. Fast 20,000 Projekte nutzen den Service von Sourceforge, etwa projektbezogene Subdomains, Mailinglisten, CVS- und FTP-Server sowie Projektbeschreibungen und -bewertungen. Um auf ein Projekt aufmerksam zu machen, ist daher der Eintrag bei Sourceforge ein wichtiger Schritt.



Abbildung 2: SourceForge ist das größte Forum für Open Source-Projekte

Bei großen Projekten mit vielen Tausend Code-Zeilen, die auf einmal veröffentlicht werden (wie bei Mozilla oder StarOffice), besteht die Gefahr, dass sich für diese Unmenge Quelltext zunächst niemand interessiert, da allein schon die Einarbeitung sehr aufwendig ist. Auch hier ist eine intensive Pflege der Community entscheidend.

Communities bei Großprojekten

Auch Open Source bedeutet Aufwand

Generell gilt also, dass Open Source allein den Hersteller oder Initiator nicht von der Mitarbeit entbindet. Langfristig können sich zwar die Entwicklungsaufwendungen verschieben, kurzfristig bedeutet aber auch ein Open Source-Projekt eine Investition, da Personal für Support-Anfragen, Web-Site-Erstellung, Dokumentation oder Wahrung der Integrität der Software bereitgestellt werden muss.

Projekte schwerer planbar

Ein weitere Gefahr bei Open Source-Projekten ist, dass sie aufgrund der Heterogenität der Mitarbeiter und deren divergierenden Interessen kaum planbar sind. Viele Hersteller gehen daher den Weg, von Zeit zu Zeit den Code des Open Source-Projektes abzuzweigen und mit eigenen Mitteln daraus ein Release zusammenzustellen.

1.4 Open Source für Anwender

Wir haben bislang vorwiegend die Sicht eines Software-Herstellers untersucht. In diesem Abschnitt soll es mehr um den Anwender gehen. Denn die meisten Firmen, die mit Computern arbeiten, stellen keine Software her, sondern nutzen sie nur für ihr Geschäft.

1.4.1 Vorteile der Anwender

Zunächst einmal interessiert es die meisten Anwender selten, ob sie die Software im Quelltext oder als Binärpaket erhalten. Viele Linux-Anwender installieren heute beispielsweise nur ihre Distribution und kompilieren nicht den Kernel. Neben den allgemeinen Vorzügen freier Software, die wir oben diskutiert haben, also höhere Sicherheit, höhere Qualität – wichtig gerade für Infrastrukturdienste – und höhere Reife, gibt es aber noch weitere Aspekte, die für Anwender interessant sind.

Unabhängigkeit von einem Hersteller

Baut ein Benutzer einen wichtigen Teil seines Geschäftes auf der Software eines bestimmten Herstellers auf, so macht er sich von diesem abhängig. Sind etwa in der nächsten Version größere Änderungen enthalten, muss der Anwender die entsprechenden Anpassungen seiner Umgebung nachvollziehen, wenn er nicht seine bisherigen Investitionen verlieren will. Dieses Problem haben derzeit viele Firmen mit der Umstellung vom NT-Domänenkonzept auf Windows 2000/Active Directory.

Verfügbarkeit bei Verschwinden des Herstellers

Besonders kritisch kann es werden, wenn der Hersteller plötzlich nicht mehr am Markt ist – weil er Konkurs gegangen ist, aufgekauft wurde usw. Hier müssen die Anwender um die Weiterexistenz ihrer Software fürchten, was beim Einsatz in kritischen Bereichen besonders ärgerlich ist.

Beide Aspekte fallen bei Open Source weg. Da es keinen direkten Hersteller gibt (höchstens einen „Projektleiter“), begibt sich der Anwender in keinerlei Abhängigkeiten. Zur Not kann er die Software selbst weiter pflegen, wenn sie für ihn einen besonderen Stellenwert hat.

Außerdem enthält Open Source-Software automatisch nur offene Standards. Damit ist stets die Interoperabilität zu anderen Anwendungen, die denselben Standard unterstützen, gewährleistet, was im Zeitalter von EAI und B2B besonders wichtig werden kann.

Offene Standards

Und gerade im Internet-Business finden sich auch die meisten professionellen Anwender von Open Source-Software. Beispielsweise setzt der größte deutsche Internet-Provider 1&1 Internet AG, der jeden Monat über 100.000 neue Domains einrichtet und hostet, komplett auf Linux, sowohl bei der eigenen Web-Site als auch bei den Web-Spaces der Kunden. Eine ähnliche Strategie hat auch der E-Mail-Anbieter GMX. Andere prominente Linux-Server sind Amazon.com sowie die Suchmaschinen Google.com oder AltaVista.

Linux im Internet

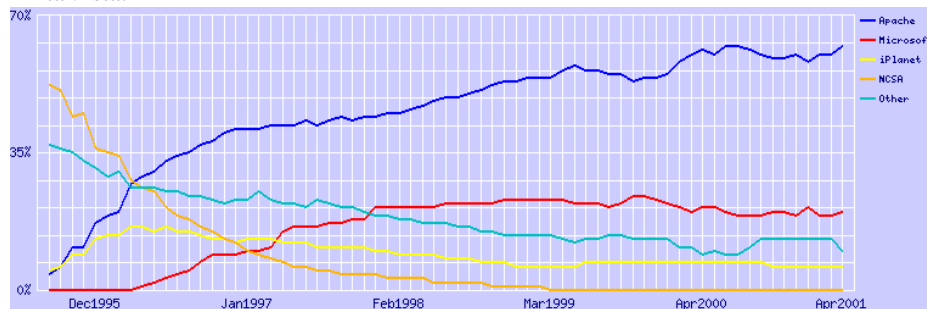


Abbildung 3: Marktanteil der führenden Web-Server über alle Domains des Internet, August 1995 bis April 2001 (nach [13])

Noch überzeugender ist der Web-Server Apache. Wie die jüngsten Zahlen von Netcraft [13] zeigen, laufen 62.5 % aller Sites des Internet mit dieser Open Source-Software. Auch große Firmen wie Oracle, Apple oder Hewlett-Packard nutzen dieses freie Programm für ihren Web-Auftritt.

Apache beliebtester Web-Server

1.4.2 Vom Konsumenten zum „Prosumenten“

Anwender von Open Source-Software geraten im Verhältnis zu traditioneller Software in eine neue Rolle.

Bei kommerziellen Applikationen verhält sich der Benutzer meist passiv: er kauft das Produkt und geht damit um. Tauchen Probleme auf, versuchen die Hersteller zwar mit unterschiedlichem Eifer zu helfen. Oftmals ist diese Hilfe aber wieder kostenpflichtig (bzw. nur über teure 0190-Nummern erreichbar) oder weitgehend unbrauchbar. Viele Anwender warten auch vergeblich auf eine Rückmeldung, wenn sie sich über

Der resignierte, passive Anwender

einen Programmfehler beschwert haben, und leben daher resigniert mit den Unzulänglichkeiten ihrer Software.

Bei Open Source-Anwendungen ist dies anders. Hierbei hat der Benutzer prinzipiell die Möglichkeit, den Quellcode zu verändern und so die Probleme zu beheben. Sicher wird dies nicht jeder Anwender tun, denn dazu gehört aufgrund der Komplexität vieler Projekte oft einige Programmiererfahrung. Aber für Firmen kann es sich durchaus lohnen, wenn die IT-Abteilungen Kapazitäten dafür bereitstellen oder einen Dienstleister beauftragen. Denn eine wichtige Anwendung, die immer wieder Probleme bereitet, kann zu erheblichem Produktivitätseinbußen führen.

Der „Prosument“

Langfristig bedeutet das aber auch, dass der Anwender zum Mitentwickler wird. Der Sozialwissenschaftler Alvin Toffler sprach in [14] schon vor fast zwanzig Jahren davon, dass zukünftig die klassische Trennung zwischen Produzent und Konsument immer weiter verwischen wird, und nannte die entstehende Rolle den „Prosumenten“. Bei Open Source ist diese Vision bereits Wirklichkeit geworden. Jeder Konsument, der das möchte, kann zum Produzenten werden und so seine eigenen Schwerpunkte einbringen. Im Grunde sollte er das sogar.

Mitarbeit als Gegenwert

Die Idee der freien Software ist es nämlich auch, dass der Benutzer für die Leistungen, die er in Form von vielen Programmen erhält, wieder etwas zurückgibt – kein Geld, wie im kommerziellen Fall, sondern eigene Arbeitszeit. Bei einem Unternehmen wäre das Arbeitszeit von Mitarbeitern. Auch wenn es gegenwärtig bereits sehr viele Open Source-Projekte gibt – auf Dauer kann diese Modell nur dann funktionieren, wenn es nicht nur immer Selbstbediener gibt, die nichts für den enormen Wert zurückgeben, den sie erhalten.

1.4.3 Probleme mit Open Source

Da hinter einer Open Source-Software nur selten eine Firma steht, darf man nicht erwarten, eine solche Anwendung in genau derselben Ausstattung zu bekommen wie eine kommerzielle. Die meisten Entwickler sind eher an Funktionalität als an Ergonomie oder umfassender Dokumentation interessiert. Insofern kann man Open Source-Applikationen als „Rohprodukte“ begreifen, die erst für den täglichen Gebrauch nutzbar gemacht werden müssen.

Support ist nötig

Das kann durch den Anwender selbst geschehen, wenn er über das nötige Know-How verfügt oder sich die Kenntnisse im Internet zusammensuchen möchte, oder über Dienstleister, sowohl firmeninterne als auch –externe. Eine Reihe von Unternehmen haben sich als Service-Anbieter für Open Source einen Namen gemacht, allen voran die Linux-Distributoren RedHat und SuSE. Wer also ernsthaft Open Source-

Anwendungen einsetzen will, sollte einen Support-Vertrag dazu abschließen, um die optimale Verfügbarkeit für alle Geschäftsfälle gewährleisten zu können. Damit ist nur vordergründig eine neue Abhängigkeit zu einem „Hersteller“ geschaffen. Denn da die Quellen offen sind, kann sich jeder Dienstleister das nötige Wissen aneignen und der Auftraggeber kann ihn jederzeit austauschen.

Eine der häufigsten Argumente der Verantwortlichen gegen Open Source war: „Wenn ich keinen Hersteller mehr habe, wen kann ich verklagen, wenn die Software fehlerhaft ist und Schäden verursacht?“ Die einfache Antwort vieler Open Source-Befürworter („Versuchen Sie doch mal, Microsoft bei Fehlern zu verklagen!“) löst das Problem nicht. In der Tat enthält etwa die GPL ausdrücklich den Hinweis, dass keinerlei Gewährleistung übernommen wird. Das ist zwar in den USA so pauschal möglich, in Deutschland ist die Rechtslage dagegen unklarer.

*Rechtliche Beziehungen
bei GPL*

Wenn Sie Open Source-Software von einem Server selbst herunterladen, muss Sie der Betreiber dieses Servers darauf aufmerksam gemacht haben, dass er für die Inhalte keine Haftung übernimmt, und auf mögliche Gefahren bei der Benutzung hingewiesen haben. Damit ist dieser Betreiber für den Inhalt nicht verantwortlich und kann auch nicht belangt werden. Bei Distributionen, die Sie käuflich erwerben, muss der Distributor Ihnen nur für die Teile Haftung gewähren, die er selbst hergestellt hat, nicht für die, die er selbst als Schenkung überlassen bekam. Für solche Software, die unentgeltlich überlassen wird, bestehen auch keine Gewährleistungsansprüche. Der Distributor muss höchstens ähnlich wie der Server-Betreiber, dass er die Software nur aus zuverlässiger Quelle erhält. Er haftet für freie Software zwar prinzipiell nur bei grober Fahrlässigkeit oder Vorsatz; allerdings haftet er auch bei leichter Fahrlässigkeit für die Auswahl, Beschaffung, Installation und Konfiguration der Software. Insgesamt muss er auf alle entgeltlichen Leistungen wie die Beschaffung, das Konfigurieren, Kompilieren und Testen der Programmpakete sowie das Herstellen der Datenträger auch Haftung gewähren. (Darstellung der Rechtslage nach [15])

Gewährleistung

Was heißt diese juristische Situation nun in der Praxis? Wenn es wirklich um geschäftskritische Anwendungen geht, sollte sich die Firma entweder selbst genügend Know-How zulegen, um die benötigte Funktionalität zu gewährleisten, oder damit einen externen Dienstleister betrauen, in dessen Vertrag dann auch Haftungsaspekte aufgenommen werden können. Vergleicht man die Situation mit kommerzieller Software, ist die Lage bei Open Source immer noch deutlich besser, da dort die AGB bzw. die Lizenzbestimmungen die Haftung oft auf den rechtlich geringst möglichen Umfang beschränken. Zudem sind nach einer Auflösung der Herstellerfirma überhaupt keine Ansprüche mehr durchsetzbar.

Lösung in der Praxis

Letztendlich muss man zu bedenken geben, dass auch bei sonstigen Fehlern (und Fehlentscheidungen) im Geschäftsleben das Unternehmen selbst dafür gerade stehen können muss. An Open Source-Software ist bislang noch keine Firma Konkurs gegangen, an Missmanagement schon Hunderttausende.

1.5 Fazit

Die Offenheit, die für Open Source sprichwörtlich ist, stellt für Hersteller wie Anwender eine enorme Chance dar. Hersteller können damit Marketing betreiben und ein positives Image in der Öffentlichkeit erzeugen, den Markt für weitere Produkte bereiten, Standards setzen oder die Pflege laufender Anwendungen an eine Community übergeben. Gerade der Aufbau und die Motivation einer solchen Community sind jedoch die Aspekte, die über Erfolg oder Misserfolg eines Open Source-Projekts entscheiden. Auch muss man anders mit Produktzyklen und Zeitplanungen umgehen, als das bei proprietären Systemen üblich ist.

Die Anwender erhalten, wenn sie bereit sind, sich darauf einzulassen, mit dem Quellcode die detaillierteste Dokumentation, die man sich wünschen kann. Abläufe lassen sich so bis auf die unterste Ebene debuggen, womit sich alle Arten von Fehlern besser aufspüren lassen. Je intensiver der Anwender mit den Quellen arbeitet, desto mehr verschwindet die Grenze zum Herstellerteam. Er erhält also eine neue Rolle, in der er aktiv auf das Produkt Einfluss nehmen kann. Die Frage der Produkthaftung wird dabei jedoch erheblich schwieriger, da einerseits das Entwicklerteam diese aus guten Gründen nicht gibt (und selbst nach deutscher Rechtslage nicht geben muss), andererseits oft von Endkunden oder Management diese oft gefordert wird. Hier ist es am sinnvollsten, entweder diese Haftung in bestimmten Grenzen selbst zu übernehmen oder einen entsprechenden Vertrag mit einem Dienstleister abzuschließen.

Auch wenn damit eine andere Denk- und Handlungsweise verbunden ist, ist Open Source ein höchst interessantes Modell. Der Erfolg von Linux, Apache, MySQL, PHP usw. haben gezeigt, dass es am Durchbruch keine Zweifel mehr geben kann, ja dass dieser gerade schon vor sich geht. Wie so oft bei Trends dürfte auch hier gelten: es hat der den meisten Erfolg, der sich frühzeitig darauf einstellt.

1.6 Literatur

[1] <http://www.fsf.org/copyleft/gpl.html>

- [2] <http://www.fsf.org/philosophy/license-list.html>
- [3] E. S. Raymond: "The Cathedral and the Bazaar", <http://tuxedo.org/~esr/writings/cathedral-bazaar/>
- [4] E. S. Raymond: „The Magic Cauldron“, <http://tuxedo.org/~esr/writings/magic-cauldron/>
- [5] B. Behlendorf: „Open Source as a Business Strategy“, <http://www.oreilly.com/catalog/opensources/book/brian.html>
- [6] J. Franz: "3D-Modellierung mit Open CASCADE", *Linux Enterprise*, 3/01, S. 28-30
- [7] C. Werry, M. Mowbray (Hrsg.): "Online Communities: Commerce, Community Action, and the Virtual University", Prentice Hall, 2000.
- [8] Sourceforge, <http://sourceforge.net>
- [9] R. Birkenmaier, O. Gräbner: „In Bed with Linux“, *Linux Enterprise*, 4/00, S. 38-43, <http://www.entwickler.com/le/ausgaben/2000/4/artikel/1/online.html>
- [10] Open Source Initiative: „The Open Source Case for Business“, http://www.opensource.org/advocacy/case_for_business.html
- [11] F. Hecker: "Setting Up Shop: The Business of Open Source Software", <http://www.hecker.org/writings/setting-up-shop.html>
- [12] T. Wieland: „Linux als Geschäftsfaktor“, *Linux Enterprise*, 2/00, S. 30-34, http://www.drwieland.de/articles/Linux_Geschaeftsfaktor.html
- [13] Netcraft: "Web Server Survey", <http://www.netcraft.com/survey/>
- [14] A. Toffler: „The Third Wave“, Morrow, 1980.
- [15] J. Siepmann: "Lizenz- und haftungsrechtliche Fragen bei der kommerziellen Nutzung Freier Software“, <http://www.kanzlei-siepmann.de/linux-tag/vortrag.html>