Open Source Jahrbuch 2004

Zwischen Softwareentwicklung und Gesellschaftsmodell

Herausgegeben von Robert A. Gehring und Bernd Lutterbeck

Open Source Jahrbuch 2004

Zwischen Softwareentwicklung und Gesellschaftsmodell

Herausgegeben und bearbeitet von:

Roman Büttner
Student der Geschichte und Informatik, Berlin

Yacine Gasmi
Student der Informatik, Berlin

Robert A. Gehring
Diplom-Informatiker, Wissenschaftlicher Mitarbeiter am Institut für Wirtschaftsinformatik der TU Berlin

Andreas John
Student der Informatik, Berlin

Svetlana Kharitoniouk
Studentin des Wirtschaftsingenieurwesens, Berlin

Bernd Lutterbeck
Professor für Informatik und Gesellschaft an der TU Berlin, Jean-Monnet-Professor für europäische Integration

Patrick Stewin
Student der Informatik, Berlin

Martin Unger
Student der Informatik, Berlin

2004

Lehmanns Media LOB.de

Bibliografische Informationen der Deutschen Bibliothek:
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über **<http://dnb.ddb.de>** abrufbar.

Open Source Jahrbuch 2004
Zwischen Softwareentwicklung und Gesellschaftsmodell
Gehring, Robert A.; Lutterbeck, Bernd (Hrsg.)

Berlin, 2004 - Lehmanns Media - LOB.de
ISBN: 3-936427-78-X

Satz und Layout: Roman Büttner, Berlin
Einbandgestaltung: Stephan Ecks, Berlin,
unter Verwendung der *GNU General Public License*
Druck und Verarbeitung: AZ Druck und Datentechnik, Kempten

Printed in the European Union.

Inhalt

Kapitel 3 – Technik

Kapitel 4 – Recht und Politik

Kapitel 5 – Gesellschaft

Vorwort der Herausgeber

Robert A. Gehring und Bernd Lutterbeck

Es ist schon erstaunlich, in welch kurzer Zeit sich das Phänomen „Open Source" weltweit ausgebreitet hat – eine ohne das Internet undenkbare Entwicklung. Man kann dies gut an einigen Arbeiten nachvollziehen, die in unserer Forschungsgruppe im Abstand von mehreren Jahren geschrieben wurden: Gehring (1996), Ardal (2000) und Leiteritz (2002). Während Gehring (1996) wohl weltweit die erste Arbeit war, die sich intensiv und grundlegend mit rechtlichen Aspekten auseinander setzte, verfolgten die beiden späteren Arbeiten eine ökonomische Zielrichtung.

Atila Ardal hat damals (2000) Neuland betreten. Es gab praktisch noch keine Literatur, die bei der Beantwortung der von ihm aufgeworfenen Fragen (im Kern: Ist Open Source betriebswirtschaftlich sinnvoll?) geholfen hätte. Er musste aus diesem Grunde noch viel mit Analogien arbeiten. Mehr als Anfänge eines theoretischen Modells konnte man da noch nicht erwarten.

Zwei Jahre später (2002) hat sich für Raphael Leiteritz die Situation schon entscheidend verändert. Zum einen waren sehr präzise Aussagen über die betriebliche Praxis von Open Source möglich: Raphael Leiteritz hatte als Gründer und späterer CEO eines über einige Jahre sehr aktiven Start-up-Unternehmens viele positive und negative Erfahrungen bei der Vermarktung von Open-Source-Software sammeln können.

Hinzu kam, dass das neue Phänomen inzwischen fachübergreifend zum Gegenstand intensiver Forschungen geworden war. Im Ergebnis lagen mittlerweile zahlreiche, teils hochwertige Beiträge aus den unterschiedlichsten Wissenschaftsdisziplinen vor – Ökonomie, Soziologie, Recht, Informatik –, die zu zahlreichen Aspekten von Open Source Stellung bezogen. Daraus wurde klar, dass es sich bei Open Source weder um eine Eintagsfliege noch um einen auf Software beschränkten Ansatz handelte.[1] Um nicht unterzugehen in dieser Vielfalt, hat Raphael Leiteritz im Rahmen seiner Diplomarbeit ein heuristisches Modell entwickelt, mit dem er alle Aspekte einigermaßen vernünftig ordnen konnte. Wir finden, es leistet auch heute noch gute Dienste bei der Orientierung:

---

[1] Das ist auch der Grund, warum wir vorrangig von Open Source sprechen: Die Bedeutung des Open-Source-Ansatzes reicht unserer Meinung nach, wiewohl aus der Software-Entwicklung hervorgegangen, weiter. Der offene Umgang mit Wissen betrifft eben mehr als nur Software. Dort wo wir spezifische Softwarefragen ansprechen, wird, wo notwendig, zwischen Freier Software und Open-Source-Software differenziert.

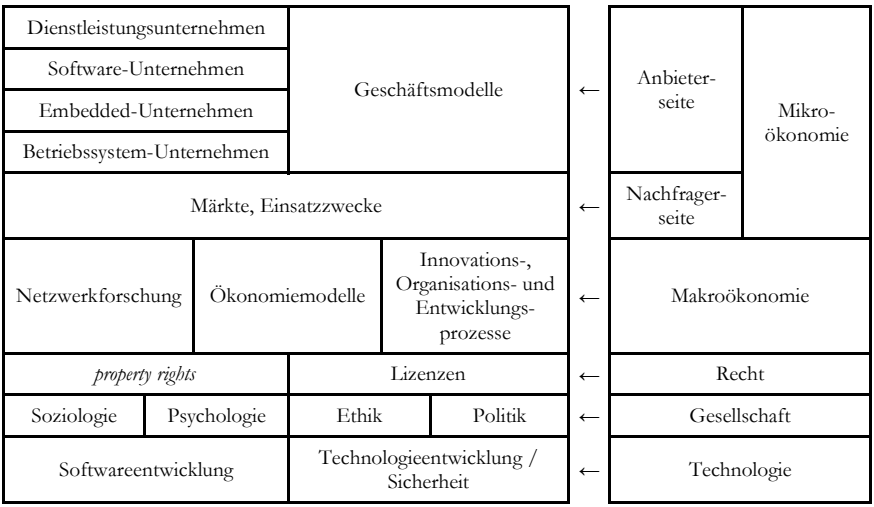


Abb. 1: Heuristisches Modell nach Leiteritz (2002)

Natürlich kann kein Wissenschaftler allein mehr diese durchaus beeindruckende Vielfalt in all ihren Facetten überblicken.[2] Von der Open-Source-Bewegung kann man aber lernen, dass man die eigenen Schwächen durch neue, andere Formen der Kooperation überwinden kann. Schon seit einigen Jahren ist deshalb Open Source fester Bestandteil unseres Lehrplanes. Das gilt sowohl für die Behandlung des Themas als auch für die Verwendung von Elementen der Open-Source-Methode in der Lehre. Ein Ergebnis dessen ist dieses Buch.

Entstanden ist die Idee zu diesem Buch vor fast drei Jahren, als die (nunmehrigen) Herausgeber diskutierten, wie man Studenten das Thema *praktisch* nahe bringen könnte. Jeweils im Sommersemester eines Jahres führen wir Praktika durch, in deren Rahmen sich unsere Studierenden mit einer Vielfalt von Problemen aus dem Bereich Informatik und Gesellschaft auseinander setzen. Ermutigt durch den überwältigenden Erfolg der WIDI–Studie[3] des Jahres 2000, einer empirischen Studie über Free- bzw. Open-Source-Softwareentwickler, beschlossen wir, ein so ambitioniertes Unternehmen in Angriff zu nehmen, wie es ein Buch ist. Orientierung bot dabei die lange Tradition an vielen Hochschulen in den USA, renommierte Fachzeitschriften durch eine Redaktion aus Studierenden betreuen zu lassen. In einer Zeit, in der Universitäten und Hochschulen finanziell ausbluten und die Kosten für Fachliteratur selbst für Hochschullehrer in unerschwingliche Höhen klettern, muss man neue Wege gehen – Open-Source-Wege. Dabei kann man, das zeigt unsere Erfahrung, auf die Unterstützung vieler unterschiedlicher Menschen bauen.

An dieser prominenten Stelle möchten wir vor allem den „Machern" dieses Jahrbuchs unsere Anerkennung aussprechen – der Redaktion. Alle sind sie ja Studierende, die fast ein ganzes Jahr lang einen Teil ihrer Zeit für das Gelingen des Projekts

2 Einen guten Überblick verschafft die Open-Source-Bibliographie O'Reilly (2002).
3 Robles, Scheider, Tretkowski und Weber (2000).

geopfert haben. Alles, was man durch irgendwelche Uni-Scheine abgelten kann, war längst geschehen. Geblieben ist ein freiwilliger Einsatz für eine sinnvolle Sache. Man weiß inzwischen sehr gut, dass intrinsische Motivation ein wesentlicher, vielleicht der wichtigste Teil des Geheimnisses von Open-Source-Projekten ist. Dieses Buch mag ein weiterer Beweis dafür sein. Profitiert hat die Gruppe ganz sicher auch vom Einsatz von Svetlana Kharitoniouk, der einzigen Frau in der Gruppe, die sich zutraute, ein solches Unternehmen zu leiten. Ihre ökonomischen Kenntnisse waren dabei erkenntlich eine gute Voraussetzung.

Bei der Auswahl der Beiträge hat die Redaktion sich – haben wir Herausgeber uns – um hohe Qualität bemüht. Das Vorschlagsrecht lag in erster Linie bei der Redaktion, die Herausgeber sind herbeigeeilt, wenn ihre Hilfe nötig war. Das Ergebnis kann sich, wie wir finden, sehen lassen. Natürlich kann man sich noch verbessern – eine Aufgabe für das nächste Jahrbuch, 2005. Die Vorbereitungen dazu haben bereits begonnen.

Wir haben Wert darauf gelegt, dass verschiedene Autorinnen und Autoren zu ähnlichen Themen zu Wort kommen. Die Vielfalt der Meinungen und das ganze Spektrum der Thesen zu erfassen war unser erstes Ziel. Das ist uns teilweise gelungen. Nicht immer hatten die Wunschautoren Zeit, und nicht immer wurden die Anfragen der Studierenden ernst genommen. So kommt es, dass in einigen Rubriken erkennbar ein Ungleichgewicht entstanden ist.

Man sollte nicht verschweigen, dass es auch Konflikte gab. Wir haben zB. einen Anwenderbericht einwerben können, der sich sehr negativ zum Einsatz von Open-Source-Software in dem entsprechenden Bereich äußert – darf man so etwas abdrucken? Das Redaktionsteam hat intensiv gerade über diesen Fall diskutiert. Das Ergebnis: Man darf nicht nur, man muss sogar! Was wäre damit gewonnen, den Lesern und Leserinnen eine heile Welt vorzuspielen? Und ist nicht gerade eine Einsicht aus der Open-Source-Praxis, Fehler oder auch nur Schwierigkeiten offen zu legen, zu diskutieren – und es beim nächsten Mal besser zu machen?

Leider ist es uns nicht gelungen, diesen prinzipiellen Gedanken der Fairness dem Unternehmen Microsoft verständlich zu machen. Auch nach zahlreichen mündlichen wie schriftlichen Anfragen auf unterschiedlichen Hierarchiestufen hat sich das Unternehmen nicht darauf einlassen können, mit einem eigenen Beitrag seine Position zu erläutern – bedauerlich. Letztendlich nachvollziehen können wir diese Haltung nicht. Verfolgt man die aktuellen Aktivitäten des Unternehmens, so hat es den Anschein, als hätte sich Microsoft entschlossen, der anschwellenden Diskussion mit Werbekampagnen und bestellten Gutachten (z.B. Koots/Langenfurth/Kalwey 2003) zu begegnen. Wir denken, dass eine große Anzahl der Beiträge dieses Jahrbuchs den Nachweis erbringt, dass die Diskussion weiter zu fassen ist, als es sich den Autoren dieses Gutachtens schon erschlossen hat.

Unser großer Dank gilt den Autorinnen und Autoren, die das manchmal penetrante Nachbohren geduldig, vielleicht auch zähneknirschend über sich haben ergehen lassen. Wenn nicht alle auf ein Honorar verzichtet hätten, gäbe es dieses Buch nicht, nicht zu diesem Preis, und es gäbe auch nicht die zugehörige Webseite, auf der die Beiträge kostenlos zugänglich gemacht werden.

Die Kooperation mit der Buchhandlung Lehmann ermöglicht es uns, selbst ohne einen Verlag im Rücken das technische Procedere von Druck und Vertrieb mit vertretbarem Aufwand abzuwickeln; auch das finanzielle Risiko ruht dergestalt nicht allein auf unseren Schultern. Der Verzicht auf den Markennamen eines renommierten Verlages gestattet es zudem, dass die Rechte an den Texten bei den Autoren verbleiben – auch das ist ein Stück mehr Open Source.

Wen wünschen wir uns als Leser?

In erster Linie natürlich all jene, die schon selbst in einem Open-Source-Prozess stecken, sei es als Softwareentwickler, Unternehmer, Wissenschaftler oder Studierender. Ihnen wollen wir ein anschauliches Koordinatensystem an die Hand geben, ihnen zeigen, dass sie Teil eines Experimentes sind, dessen Größe zu erkennen das Alltagsgeschäft nicht immer gestattet.

Aus zahlreichen Gesprächen mit Berliner Politikern und Mitarbeitern aus Ministerien wissen wir auch, dass dort ein Bedarf an verlässlicher Information und Orientierung im Chaos besteht – vor allem bei denjenigen, die über den „Einkauf" und den Einsatz von Software und Systemen entscheiden müssen. Dieses Jahrbuch soll deshalb – auch anhand von Praktikern aus Wirtschaft und Verwaltung – einen kompakten Überblick über die intellektuelle und technologische Vielfalt des Phänomens Open Source bieten.

Und zu guter Letzt möchten wir den Skeptikern der Open-Source-Methode eine Anlaufstelle zur Prüfung ihrer Positionen geben. Auch ihre substantiierten Argumente sind zu berücksichtigen.

Open Source lebt vom „free flow of information". Vielleicht ist ja der offene „marketplace of ideas", das „knowledge commons", ein besseres Paradigma für den Aufbau der Informationsgesellschaft als der eingezäunte „marketplace of proprietary information goods"? Dieses Buch wird die Frage nicht beantworten, aber es zeigt ihre Berechtigung.

Berlin, im Januar 2004
Robert A. Gehring
Bernd Lutterbeck

Referenzen

Ardal, Atila: *Open Source – das Beispiel Linux: Ökonomische Analyse und Entwicklungsmodell eines erfolgreichen Betriebssystems*, Diplomarbeit am Fachbereich Informatik der TU Berlin, April 2000,
online http://ig.cs.tu-berlin.de/forschung/OpenSource/index.html.

Gehring, Robert A.: *Freeware, Shareware und Public Domain.* Studienarbeit am Fachbereich Informatik der TU Berlin, Juni 1996,
online http://ig.cs.tu-berlin.de/forschung/OpenSource/index.html.

Kooths, Stefan / Langenfurth, Markus / Kalwey, Nadine (2003): *Open-Source-Software. Eine volkswirtschaftliche Bewertung*, MICE Economic Research Studies, Vol. 4. Münster, Dezember 2003,
online http://mice.uni-muenster.de.

O'Reilly (2002): *Open Source Bibliography. Third edition*, O'Reilly & Associates Inc., Sebastopol 2002.

Robles, Gregorio / Scheider, Hendrik / Tretkowski, Ingo / Weber, Niels (2000): *WIDI: Who Is Doing It? A research on Libre Software developers*, Forschungsbericht, Fachbereich Informatik der TU Berlin,
online http://ig.cs.tu-berlin.de/forschung/OpenSource/index.html.

Leiteritz, Raphael (2002): *Der kommerzielle Einsatz von Open-Source-Software und kommerzielle Open-Source-Geschäftsmodelle. Zur Bedeutung von Open-Source-Software in Unternehmen und als Grundlage für Geschäftsmodelle*, Diplomarbeit am Fachbereich Informatik der TU Berlin, Mai 2002,
online http://ig.cs.tu-berlin.de/forschung/OpenSource/index.html.

Kapitel 1

Grundlagen und Erfahrungen

Einleitung

SVETLANA KHARITONIOUK UND PATRICK STEWIN

> Was in der großen Weltpolitik der Fall der Berliner Mauer war, das wird dieses Votum in unserer Branche sein.
>
> RICHARD SEIBT, Vorstandsvorsitzender der Suse Linux AG[1]

Der Beschluss der Münchener Regierung im Mai letzen Jahres, 16000 EDV-Arbeitsplätze der Stadtverwaltung von der marktbeherrschenden Microsoft-Software auf Linux und andere Open-Source-Produkte umzustellen, hat für viel Aufregung in der Öffentlichkeit gesorgt. „Eine Entscheidung Münchens für Linux hätte einen besonderen Leuchtturmcharakter für andere Kommunen", betonte Andreas Gebhard von der Initiative Bundestux (Bundestux 2003), „weil hier erstmals die gesamte Verwaltung einer Millionenstadt auf Freie Software umgestellt würde."

Die Schlagzeilen um dieses Ereignis aus dem Sommer 2003[2] haben wieder einmal verdeutlicht, dass Open Source und Freie Software nicht nur ein Thema für Informatiker, sondern auch ein Gegenstand der Politik sind. Die politischen Aspekte von Open Source (OS) werden von Bundestux – der Initiative für die Einführung von Freier Software im Deutschen Bundestag – wie folgt formuliert: *Ordnungspolitik und Freier Wettbewerb*, *Demokratie und Standortvorteile* (Bundestux 2003).

Für den durchschnittlichen PC-Benutzer gehören solche Aspekte nicht zum Alltag. Open-Source-Software (OSS), wie z.B. Linux, Openoffice oder Mozilla, kennt er meistens nur vom Namen und nicht durch die Benutzung auf dem eigenen PC. In der Wirtschaft und im öffentlichen Bereich sieht das inzwischen anders aus. Dort wird das Interesse auf OSS unter anderem durch versprochene Kostenvorteile und Stabilität gelenkt.[3] Auch in der Wissenschaft spielt das Phänomen Open Source eine

---

1 In (Kerbusk 2003).

2 Weitere Informationen zur Münchner Entscheidung unter: http://www.lochner-fischer.de/frameset.html?themel/linux3.htm oder http://linux-muenchen.de/.

3 Studien dazu sind unter http://www.dbresearch.de/PROD/PROD0000000000047532.pdf, http://www.it-surveys.de/itsurvey/pages/studie_oss_executive_summary.html oder http://www.pro-linux.de/news/2003/5265.html zu finden.

wichtige Rolle. Für die aus den verschiedensten Bereichen stammenden Forscher wirft das Thema um die Open-Source-Bewegung viele Fragen auf:

- Was bringen offene Standards in der IT, und ist in diesem Zusammenhang OS sicher?
- Was motiviert viele Entwickler, sich unentgeltlich in ihrer Freizeit in der OS-Gemeinde zu engagieren?
- Welche Vorteile haben OS-Lizenzen für den Nutzer bzw. den Anbieter und Entwickler?
- Welche Vorteile haben öffentliche Institutionen bei der Nutzung von OSS?
- Soll OS-Nutzung und -Entwicklung (staatlich oder betrieblich) gefördert werden?
- Kann man mit OS Gewinne erwirtschaften?
- Können Information und Software Eigentum sein, oder sollten sie generell als „frei" gelten?
- Ist OS-Entwicklung ein Vorbild für die Forschung?
- Ist Open Source ein besseres Gesellschaftsmodell?

Die vorliegende Publikation will – nicht zuletzt, um die Entscheidung des Lesers für oder wider Open-Source-Software zu erleichtern – Einblicke in derzeitige Diskussionen bieten. Aktuelle Entwicklungen werden in diesem ersten Band eines als Reihe angelegten Jahrbuches zunächst mit dem Focus auf die Bundesrepublik Deutschland in ausgesuchten Beiträgen verschiedener Autoren präsentiert. Die nach Themenschwerpunkten sortierten Kapitel dieses Bandes werden jeweils durch eine Einleitung ergänzt, die einen kurzen Überblick verschaffen soll.

Im ersten Kapitel wird auf die Grundlagen von Open Source und Freier Software eingegangen. Es werden die Geschichte, die Definition und die bekanntesten bzw. erfolgreichsten Projekte vorgestellt. In den anschließenden Artikeln berichten Dienstleister und Anwender über ihre Erfahrungen bei der Migration von Microsoft-Produkten zu Linux und anderer Open-Source-Software. Den Erfahrungsberichten folgt eine Beschreibung der KDE-Gemeinschaft, die von der beteiligten Projektleiterin Eva Brucherseifer verfasst wurde. Seinen Abschluss findet dieses Kapitel in einer Rezension von Steven Webers grundlegendem Buch[4] „Success of Open Source".

Im Folgenden soll zunächst der Begriff Open Source bezogen auf Software eingeordnet werden.

Der Open-Source-Begriff

Der Begriff Open Source (OS) wird in diesem Buch, wie oft auch in anderer Fachliteratur und in der Presse, für Freie Software (*free software*) und *open source software* verwendet. Die Unterschiede dieser beiden Ausprägungen sollen im Laufe der folgenden Abschnitte geklärt werden.

Open Source heißt aus dem Englischen übersetzt „quelloffen". Um das zu erläutern, muss zunächst näher auf die Entstehung von Software eingegangen werden. Jede Software wird von einem Programmierer in einer Programmiersprache ge-

[4] Erscheint im April 2004.

schrieben und kann anhand des so erzeugten Programmtextes vom Menschen nachvollzogen werden. Wird dieses als Quelltext, Quellcode oder Sourcecode bezeichnete Produkt mit oder ohne das dazugehörige ausführbare Programm veröffentlicht (und durch bestimmte Lizenzen in seiner freien Nutzung gesichert), spricht man von Open-Source-Software.

Damit ein Programm gestartet werden kann, wird der Programmtext in eine für den Computer verständliche Form (die für den Menschen nicht mehr lesbar ist) übersetzt (kompiliert), die dann von der Maschine ausgeführt werden kann. Bei handelsüblicher – also meist proprietärer – Software bleibt wegen nachvollziehbarer Interessen des anbietenden Unternehmens der Quellcode verborgen. Der Benutzer erhält ein Programm nur in der für den Computer ausführbaren Form, er kann damit zwar die Funktion, nicht aber die Art des Funktionierens seiner Erwerbung nachvollziehen. Entsprechende Lizenzbedingungen schützen das fertige Produkt vor dem Vervielfältigen, Verändern, Weiterverbreiten und Dekompilieren, also dem Rückübersetzen in eine zumindest dem Experten verständliche Form. Der Nutzer sieht nur das, was auf seinem Bildschirm passiert. Ein großer Nachteil für ihn ist, dass er diese Software bei Bedarf nicht an seine konkreten individuellen Bedürfnisse anpassen kann. Jede Weiterentwicklung und Fehlerbehebung liegt in der Hand des Herstellers.

Diese Nichtflexibilität proprietärer Software hat viele Entwickler bewegt, eigene Programme zu schreiben und diese mit ihrem Quellcode im Internet zu veröffentlichen. Als Alternative zur proprietären Software ist eine „freie“ bzw. „quelloffene“ Software entstanden, die jeder nach seinen Vorstellungen anpassen kann. Lizenzgebühren werden nicht erhoben, im Allgemeinen kann jeder Nutzer den Code aus dem Internet herunterladen. Verbesserungsvorschläge und Änderungen kann er ebenfalls im Internet, z.B. durch Mailinglisten, veröffentlichen. So wird freie bzw. offene Software durch Anregungen der Nutzer und weltweites Mitwirken anderer Entwickler ständig verbessert. Open-Source-Software wird deshalb inzwischen nicht nur als Produkt, sondern auch als Entwicklungsmodell begriffen. Beachtenswert ist dabei, dass die Entwicklergemeinde durch die schnellen Kommunikationswege und offen gelegte Ergebnisse oft viel schneller als ein Unternehmen reagieren kann.

Der Entwicklungsprozess wird von den Beteiligten selbst koordiniert, legt aber keine Beschränkungen für den Einzelnen fest. Das Phänomen der freien Software besteht auch darin, dass eine große Anzahl der Entwickler in ihrer Freizeit und meist kostenlos in den Projekten arbeitet. Linus Torvalds, der Initiator von Linux, verdiente seinen Unterhalt lange Zeit in einem Unternehmen, in dem er mit Linux nichts zu tun hatte. Über die Motivation von Entwicklern und die Abläufe in einer solchen Entwicklungsgemeinde erfährt der Leser mehr im Kapitel Technik und im Bericht der KDE-Projektleiterin Eva Brucherseifer in diesem Kapitel des Buches.

Entscheidende Einflüsse auf die Entstehung der Open-Source-Bewegung

Um die Entstehung der OS-Bewegung klären zu können, muss die Geschichte des Internets und der Software herangezogen werden.

Dieses Netz ist nicht nur Produkt einer von vielen Beteiligten geschaffenen wissenschaftlichen Arbeit, sondern auch die Entwicklungsumgebung für Informatiker und die Kommunikationsgrundlage vieler anderer Wissenschaften.[5]

Die Koordination der Entwicklungsarbeit für das Internet bzw. der technischen Standards hat 1986 die Internet Engineering Task Force (IETF) übernommen.[6] Die Standards wurden von Anfang an offen gelegt. Jeder konnte sie einsehen, sich bei den Diskussionen auf den Mailinglisten beteiligen und somit Verbesserungsvorschläge und Kommentare einarbeiten. Es entstanden schnell weitere Diskussionsmöglichkeiten in öffentlichen Foren und durch synchrone Kommunikationsformate (wie z.B. Chat).

Das „Entwicklungsmodell" des Internets ähnelt dem der OSS. Einmal vorhanden, ermöglichte das Internet dann eine weltweite, kostengünstige und verwaltungsarme Zusammenarbeit auf elektronischem Wege. Damit schafft es eine wichtige Grundlage für die OS-Gemeinschaft und für die Verbreitung Freier Software.

Wenn Open-Source-Software im großen Stil erst durch die weltweite Vernetzung möglich wurde, so hat sie wiederum auch die Entwicklung des Internets beeinflusst, wie beispielsweise durch die Integration der gängigen Internet-Protokolle in das freie Betriebssystem BSD-UNIX. Auch Projekte wie BIND und Sendmail sind in diesem Zusammenhang zu erwähnen. Die Entwicklung des Internets hat Open-Source-Software entscheidend vorangetrieben, während durch diese wiederum das weltweite Netz entscheidende Verbesserungen erfuhr.

Im Gegensatz zum Internet wurde die Geschichte von Open-Source-Software aber nicht nur durch die öffentliche Entwicklung (z.B. an Universitäten), sondern auch durch das Engagement einzelner Wissenschaftler und den kommerziellen Kommunikations- und Computermarkt geprägt.

Die ersten Computernutzer waren Fachleute, welche ihre Software selbst nachvollziehen und weiterentwickeln konnten. Software wurde in der Anfangszeit der Computer als „Bedienungsanleitung" kostenlos mit der Hardware ausgeliefert. Ende der 60er Jahre begann der damalige Marktführer IBM, den zum Bedienen eines Computers erforderlichen Programmcode separat anzubieten, und schuf so nach und nach einen Markt für proprietäre Software.[7] Die Standardisierung der Hardware – wie z.B. durch den IBM-PC, der bis heute die Mehrzahl der Schreibtische bevölkert – ließ den Computer dann schnell zu erschwinglicher Massenware werden und führte in der Folge zu einer enormen Nachfrage an Software. Immer mehr Unternehmen, Behörden und schließlich Privathaushalte nutzten die Computertechnik. Als der Benutzer nicht mehr automatisch gleichzeitig Computerfachmann war, hörte er auf, sich für die genauen Funktionen „seiner" Programme zu interessieren. Er wollte sie benutzen. Andere Nutzer entwickelten hingegen ein starkes Interesse für Software und wurden zu Programmierern – gerade auch zu OS-Programmierern, weil ihnen die Funktionen der als „Blackbox" ohne Quellcode ausgelieferten proprietären Software nicht ausreichten, weil sie verstehen wollten, was ihr Rechner ge-

5 Vgl. (Grassmuck 2002, S. 179 ff.).
6 Siehe http://www.ietf.org.
7 Vgl. http://www-1.ibm.com/ibm/history/history/decade_1960.html.

nau tut, weil sie die Programme verändern, verbessern und an spezielle Bedürfnisse anpassen wollten.

Von UNIX zu GNU/Linux

Als Ursprung der ersten OSS wird oft das Betriebssystem UNIX genannt. Die Begriffe „Free Software" und „Open Source" sind erst viel später entstanden. Im Jahr 1969 wurde die erste UNIX-Version bei AT&T Bell Telephone Laboratories entwickelt. UNIX konnte damals von AT&T auf Grund eines Kartellrechtsprozesses nicht kommerziell vertrieben werden. Es wurde gegen geringen Lizenzgebühren an Universitäten verkauft. Da kein Support von AT&T angeboten wurde, entwickelten die Nutzer das Betriebssystem selbst weiter. Dabei kommunizierten sie untereinander schon auf elektronischem Wege per E-Mail, Newsletter, Konferenzen und Dateiübertragung innerhalb einer universitätsinternen UNIX-c*ommunity*.

An der University of Berkeley in Kalifornien wurden mehrere weiterentwickelte Programme zur Berkeley Software Distribution (BSD) zusammengestellt, deren Nachfolger FreeBSD[8] noch heute als OS-Projekt existiert. Ab 1982 wurde UNIX dann aber stärker kommerzialisiert. Der Lizenzdruck von AT&T erschwerte eine offene Weiterentwicklung.

Mitte der 80er Jahre wurde der Begriff „Freie Software" von Richard Stallman geprägt. Die Ursprünge seiner Idee liegen allerdings in den 70er Jahren. Als Mitarbeiter am MIT (Massachusetts Institute of Technology) lernte er 1971 im Artificial Intelligence Lab (AI Lab) die „software sharing community"[9] kennen. Diese c*ommunity* bestand aus Programmierern, die ihre Software offen gehalten, geteilt und gegenseitig verbessert haben. Sie gilt als erste Verkörperung der Philosophie der Freien Software. Als diese Gemeinschaft Anfang der 80er Jahre beinahe aufgelöst wurde, beschloss Stallman, die Voraussetzungen für eine neue Gemeinschaft zu schaffen:

> Die Antwort war klar: zuerst wurde ein Betriebssystem gebraucht. Das war der entscheidende Punkt um anzufangen, einen Computer zu nutzen. Mit einem Betriebssystem kann man viele Dinge machen; ohne kann man den Computer überhaupt nicht nutzen. Mit einem freien Betriebssystem könnten wir wieder eine neue Gemeinschaft von zusammenarbeitenden Hackern haben – und jeden einladen, sich uns anzuschließen. (Stallman 2003)

Das neue Betriebssystem GNU (GNU's Not Unix) adaptierte die Funktionalität von UNIX, wurde aber von der ersten Codezeile an neu geschrieben. Im Gegensatz zu UNIX war es „frei". Richard Stallman veröffentlichte diese Idee 1983 bei den UNIX-Newsgroups und lud zur Mitarbeit ein. Mit der Gründung der *Free Software Foundation*[10] im Jahr 1985 schuf er eine Finanzierungsmöglichkeit und einen Organisationsrahmen für den GNU-Vertrieb sowie eine Plattform für die weitere Unterstützung von Freier Software.

---

8 Siehe auch http://www.freebsd.org.
9 Vgl. hierzu (Levy 1984).
10 Vgl. http://www.fsf.org.

„‚Freie Software' hat etwas mit Freiheit zu tun, nicht mit dem Preis. Um das Konzept zu verstehen, ist an ‚frei' wie in ‚freier Rede', und nicht wie in ‚Freibier' zu denken." (Free Software Foundation 2002). Die „Freiheit" der Software wurde in der so genannten *GNU General Public License* (GPL)[11] manifestiert und setzt Folgendes voraus[12]:

- den Zugang zum Quellcode,
- die Freiheit, die Software zu kopieren und weiterzugeben,
- die Freiheit, das Programm zu ändern, und
- die Freiheit, das Programm unter denselben Bedingungen zu verbreiten.

Das GNU-Projekt hatte im Laufe der Zeit große Fortschritte gemacht und viele Programme als Freie Software in das zu vollendende Betriebssystem implementiert. Allerdings gelang es bis Anfang der 90er Jahre nicht, einen Kernel (essenzieller Kernteil eines Betriebssystems – verbirgt die direkte Ansteuerung der Hardware vor dem Benutzer) für GNU zu entwickeln.

Glücklicherweise kodierte 1991 der aus Finnland stammende Student Linus Torvalds den an UNIX angelehnten Kernel Linux, der von nun an zusammen mit der GNU-Software der FSF ein vollständiges, freies Betriebssystem (GNU/Linux) bildete. Mit diesem Zusammenschluss war ein ernst zu nehmender Konkurrent für Microsoft-Produkte und andere proprietäre Software geschaffen worden. GNU/Linux verkörpert all die Vorteile, die OSS zugeschrieben werden. Fehlende Lizenzgebühren, Sicherheit, Stabilität und Flexibilität erhöhten den Bekanntheitsgrad von OSS und haben somit zu einer weiten Verbreitung dieser alternativen Programme in der Wirtschaft und im öffentlichen Sektor beigetragen.

Zurzeit wird der Marktanteil von GNU/Linux im Serverbereich weltweit auf Platz zwei mit 26 %[13] geschätzt – die Installationen nehmen weiter zu. Verschiedene Distributoren wie SuSE, Redhat, Debian, Mandrake u.a.[14] vertreiben das Softwarepaket, sie verlangen Gebühren für Support, Schulungen, Handbücher etc., wodurch das GNU/Linux-Betriebssystem nicht mehr grundsätzlich kostenfrei ist. Die Möglichkeit, eine solche Distribution unter Verzicht auf weiteren Service gebührenfrei aus dem Internet herunterzuladen, geht dadurch aber nicht verloren.

Die Open-Source-Initiative

Im Laufe der Zeit haben als „Freie Software" veröffentlichte Programme wie GNU/Linux immer mehr an Bedeutung gewonnen. Neben der *Free Software Foundation* etablierte sich Ende der neunziger Jahre eine zweite, von Bruce Perens und Eric S. Raymond gegründete Bewegung, die sich den Namen *Open Source Initiative* (OSI) gab.

Die Gründung dieser Initiative 1998 ist insbesondere auf zwei einschneidende Ereignisse zurückzuführen: Die Firma Netscape hatte den Quellcode seines Webbrowsers zur offenen Programmierung freigegeben, damit war zum ersten Mal die offene Entwicklung eines auch bei Privatanwendern bekannten und weit verbreite-

11 Nachzulesen hier: http://www.gnu.org/licenses/gpl.html.
12 Vgl. (Grassmuck 2002).
13 Quelle: ICD, gefunden in (Kerbusk 2003).
14 Vgl. http://www.linux.org/dist/list.html.

ten Programms möglich. Weiterhin fand am 3. Februar 1998 das erste so genannte „Open-Source-Gipfeltreffen“ statt. Auf dieser Konferenz wurde das neue Label „Open Source“ gegründet (*Open Source Certification*, *Open Source Trademark*). Kurz nach diesem Treffen entstand die Webseite *www.opensource.org*, die bis heute den Internetauftritt der Open-Source-Initiative darstellt, die sich dort vorstellt und die aktuellsten Neuigkeiten zum Thema präsentiert. Weiterhin finden sich dort die Open-Source-Definition und andere interessante Dokumente wie z.B. der Aufsatz „Advocacy“[15], der weitere Hintergründe zu erläutern versucht.

Ziel der Open-Source-Initiative ist es, Open-Source-Software der Wirtschaft näher zu bringen. Die Zusammenarbeit wird gesucht, um die OS-Bewegung zu stärken und um Geschäftsmodelle zu finden, mit denen es möglich wird, mit OSS Gewinne zu erwirtschaften. An dieser Stelle wird auch deutlich, warum man den Begriff „Open Source“ eingeführt und nicht bei „Free Software“ geblieben war. Die *Free Software Foundation* verfolgt einen eher idealistischen Ansatz. Sie verkörpert eine besondere Philosophie der Freiheit: Nicht nur der Quellcode einer Software muss frei sein, sondern auch an ihm getätigte Modifikationen. Mit diesem Ansatz blieb quelloffene Software von der Wirtschaft und von den Unternehmen weitestgehend ungenutzt. Zum einen kam es zu dem Missverständnis, „free“ stünde für kostenlos und es ließe sich so mit quelloffener Software in keiner Weise Gewinn erwirtschaften. Zum anderen hegten Unternehmen die Befürchtung, dass durch die Verwendung von quelloffener Software ihre eigene „infiziert“ würde, sodass diese dann auch „frei“ sein muss.

Durch die initiierten Veränderungen der OSI erlangte Open-Source-Software und die im folgenden Absatz erläuterte Open-Source-Definition weiter an Bedeutung – unter anderem auch für Wirtschaft und Unternehmen. Ein Jahr später kam es zu einem weiteren Gipfeltreffen. Dort waren nun auch namhafte Unternehmen wie Sun Microsystems, IBM und Oracle vertreten.

Die Open-Source-Definition

Die OSI entwarf die bereits oben genannte Open-Source-Definition[16], die derzeit (24.1.2004) in der Version 1.9 existiert und die Grundlage für alle Open-Source-Lizenzen bildet.

Beim Entwurf der Definition hat man sich auf vier Schwerpunkte geeinigt, die eine konforme Lizenz beinhalten muss: *Verbreitung* (bezieht sich auf die Lizenz und die Software als solche; Punkt 1 und 7), *Schutz des Quellcodes* (Punkte 2, 3, 4), *Ausschlussmöglichkeiten* (z.B. von Personen oder Technologien; Punkte 5, 6, 8 und 10) und *Einflussnahme* (auf andere Software; Punkt 9).

Konkret beinhaltet die Definition Folgendes[17]:

1. Freie Weiterverbreitung

Es soll gewährleistet werden, dass beliebig viele Kopien der Software (auch Distributionen, also eine Zusammensetzung von mehreren Softwaremodulen) angefer-

15 Vgl. http://www.opensource.org/advocacy/.
16 Vgl. (Open Source Initiative 2004).
17 Vgl. (O’Reilly 1999).

tigt werden können. Diese Kopien dürfen weitergegeben und sogar verkauft werden. Das Verkaufen bzw. Kopieren ist dann allerdings als Dienstleistung zu betrachten, da für die Software selbst keine Lizenzgebühr erhoben werden darf.

2. Quellcode

Der Quellcode soll offen und in einer für den Menschen nachvollziehbaren Form verbreitet werden (ein Programmierer soll den Code entsprechend verändern können). Wird der Quellcode nicht zusammen mit der kompilierten Form verbreitet, dann muss klargestellt sein, dass er lizenzgebührenfrei aus dem Internet bezogen werden kann.

3. Auf dem Programm basierende Werke

Auf Open-Source-Programmen basierende Werke *sollen* unter den gleichen Lizenzbedingungen weiterverbreitet werden. Das heißt aber auch, dass eine Weiterverbreitung unter anderen Lizenzbedingungen nicht ausgeschlossen wird. Es ist also möglich, dass ein auf einem Open-Source-Programm nach diesen Regeln basierendes Werk nicht frei sein *muss*. Es gibt Open-Source-Lizenzen, die entsprechend ausformuliert wurden (wie die *Berkeley Software Distribution License*).

4. Die Unversehrtheit des Originalcodes

Originalcode und veränderter Code müssen unterscheidbar sein. Der Ruf des Autors soll dadurch erhalten bleiben, und die Nutzer sollen erkennen, wer welchen Code wirklich geschrieben hat.

5. Keine Diskriminierung von einzelnen Personen oder Gruppen

Niemand kann von der Nutzung oder Weiterentwicklung von Open-Source-Software ausgeschlossen werden. Dies würde der Idee von Open Source widersprechen. Weiterhin wird das Ziel verfolgt, möglichst viele Personen bzw. Gruppen für die Entwicklung von Open-Source-Software zu gewinnen.

6. Keine Einschränkungen für bestimmte Anwendungsbereiche

Dieser Punkt der Definition lehnt an Punkt 5 an. Allerdings zielt er auf die Anwendungsbereiche von Open-Source-Software ab. Eine mögliche Lizenz darf keine bestimmten Einsatzgebiete der Software verbieten (sonst ist sie nicht mehr Open-Source-konform). Dazu zählt auch die kommerzielle Nutzung.

7. Verbreitung der Lizenz

Es ist zu verhindern, dass Open-Source-Software auf indirektem Weg ihren Status verliert. Die Rechte des Nutzers müssen erhalten bleiben, ohne dass zusätzliche Bedingungen oder Einverständniserklärungen beachtet werden müssen.

8. Die Lizenz darf nicht für ein bestimmtes Produkt gelten

Es soll ausgeschlossen werden, dass die betrachtete Software nur innerhalb einer bestimmten Software-Distribution als Open-Source-Software gelten darf. Wird die betrachtete Software aus einer Distribution herausgenommen, so gelten weiterhin die Rechte für den Benutzer, welche der Software innerhalb der Distribution gegeben waren.

9. Die Lizenz darf andere Software nicht beeinträchtigen

An dieser Stelle der Definition wird die Einflussnahme einer Lizenz für Open-Source-Software auf andere Software thematisiert. „Die Lizenz darf andere Software nicht beeinträchtigen" besagt, dass es beispielsweise nicht möglich sein darf zu fordern, dass die gesamte Software, die auf einem Computer oder System genutzt wird, Open-Source-Software sein muss.

10. Die Lizenz muss technologisch neutral sein

Wenn eine Lizenz formuliert wird, dann muss auch darauf geachtet werden, dass die technologische Neutralität gewahrt wird. Es soll nicht möglich sein, innerhalb einer Lizenz eine bestimmte Technologie oder eine Schnittstelle vorzuschreiben und somit andere auszuschließen.

Eine Lizenz, welche als Open-Source-konform gelten soll, muss der Definition genügen. Die *GNU General Public License*, die *Berkeley Software Distribution License* und die *Mozilla Public License* sollen hier als die bekanntesten Beispiele für Open-Source-konforme Lizenzen genannt werden. Open-Source-Projekte, bei denen unter anderem diese Lizenzen angewendet werden, sind weiter unten in dieser Einleitung aufgezählt. Eine detailliertere Betrachtung von Open-Source-Lizenzen findet der Leser im Kapitel Recht und Politik.

Es ist zu beachten, dass einige dieser Lizenzen, wie die erste Lizenz für quelloffene Software, die *GNU General Public License* (GPL), schon vor der Einführung der Open-Source-Definition existierten. Die GPL und andere Lizenzen der *Free Software Foundation* entsprechen den Bedingungen des OSI, auf diesen beruhende neuere Lizenzen aber nicht unbedingt den Ansprüchen der FSF. Damit ist „Free Software" zwar immer auch „Open-Source-Software", andersherum gilt diese Beziehung aber nicht. Die Regeln für Free-Software-Lizenzen sind strenger als die für Open-Source-Software. So *müssen* beispielsweise Programme, die Code verwenden, der unter den Bedingungen der FSF lizenziert wurde, unbedingt wieder Free Software sein, wurde Code aus einem Open-Source-Produkt verwendet, *kann* das Endprodukt wiederum als Open Source vertrieben werden (dies wäre im Sinne des OSI sicher auch wünschenswert), es *muss* aber nicht. Wegen dieser asynchronen Wechselwirkung (und weil die offenen Quellen in jedem Falle der gemeinsame Nenner sind) hat sich im allgemeinen Sprachgebrauch der Begriff „Open-Source-Software" für jede Art von freier oder quelloffener Software durchgesetzt.

Die Anhänger der Freien Software vertreten weit höhere Ideale, sie propagieren eine gesellschaftliche Utopie, während die Akteure der Open-Source-Initiative vor allem das technische Entwicklungsmodell von Software mit offenen Quellen praktizieren – nach Möglichkeit zusammen mit proprietären Unternehmen. Wegen ihrer schon in den Grundlagen verschiedenen Ansätze widersprechen sich diese beiden Ideen damit nicht so sehr, wie es auf den ersten Blick scheint.

Als Open Source werden inzwischen auch Projekte, Produkte und Vorgehensweisen bezeichnet, die nichts mehr mit Software zu tun haben, aber den Definitionen und dem OS-Geist entsprechen. Beispiele dafür sind die *Open Web School*[18] – ein

[18] Vgl. http://www.openwebschool.de.

Web-Portal, wo Unterrichtsmaterialien für Lehrer und Schüler unter der GPL angeboten werden – und die von interessierten Laien im Internet erstellte Enzyklopädie Wikipedia[19], die unter der ebenfalls von der FSF erstellten *GNU Free Documentation License* erschienen ist. Es handelt sich in beiden Fällen nicht um Software im klassischen Sinne, sondern um vielfältige Inhalte, die von ihren Autoren zum Kopieren, zur Weitergabe, zum Verändern und zur Verbreitung unter den Bedingungen der „Offenheit" zur Verfügung gestellt wurden.

Vergleich von Softwaremodellen

Damit quelloffene Software richtig eingeordnet werden kann, muss neben der Darstellung ihrer Definitionen ein Vergleich mit anderen Modellen vorgenommen werden. Dass Lizenzen der üblichen proprietären Software jeden erdenklichen Umgang mit ihr, außer der Benutzung nach dem Entrichten der Lizenzgebühr, meist verbieten, aus nahe liegenden Gründen aber zumindest das Weitergeben von Kopien und das Dekompilieren, also das Rückübersetzen des gelieferten Programmcodes in eine Programmiersprache, ausschließen, dürfte im Allgemeinen bekannt sein. Daneben gibt es aber noch die auch als proprietär zu bezeichnenden Modelle Freeware und Shareware sowie den Sonderfall Public Domain.

Zum Vergleich bzw. zur Abgrenzung soll die folgende Tabelle dienen[20]. Sie enthält eine Kurzbeschreibung der Modelle und stellt wichtige Nutzungsrechte (I bis V) gegenüber:

I. Der Quellcode ist modifizierbar.

II. Modifikationen müssen immer unter denselben Bedingungen veröffentlicht werden.

III. Das Produkt ist lizenzgebührenfrei.

IV. Die Nutzung ist uneingeschränkt.

V. Die Weiterverteilung ist erlaubt.

	Kurzbeschreibung	*I*	*II*	*III*	*IV*	*V*
Free Software	Der Quellcode ist offen, und seine Modifikationen müssen auch offen bleiben.	x	x	x	x	x
Open-Source-Software	Quelloffene Software soll Unternehmen und Wirtschaft näher gebracht werden. Die kommerzielle Nutzung soll einfacher sein (im Vergleich zu Free Software).	x		x	x	x
Public Domain	Diese Software ist als ein Sonderfall zu betrachten: Der Urheber verzichtet komplett auf das *copyright*. Somit wird diese Software zum Gemeingut und kann uneingeschränkt genutzt werden. Sollte der Quellcode zur Verfügung stehen, liegt Open-Source-Software vor.	x		x	x	x

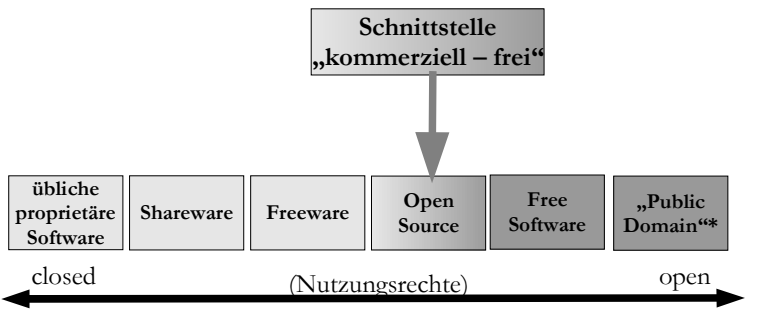
[19] Vgl. http://de.wikipedia.org.

[20] In Anlehnung an (Ploch u.a. 2002, S. 20 ff.).

	Kurzbeschreibung	*I*	*II*	*III*	*IV*	*V*
Freeware	Diese Art der Software ist keine Free Software. Es werden zwar keine Lizenzgebühren erhoben, aber der Quellcode steht nicht zur Verfügung.			x	x	x
Shareware	Hierunter wird Software verstanden, die für eine vom Autor festgelegte Testphase genutzt werden darf. Ist die Testphase abgelaufen, so sind Lizenzgebühren zu bezahlen.					x
Übliche proprietäre Software	Proprietäre Software wird als „Gegenmodell" zur Free bzw. Open-Source-Software gesehen: Die Weiterverteilung ist verboten, der Quellcode bleibt verborgen, sämtliche Modifikationen sind verboten, und es werden immer Lizenzgebühren verlangt. Es bleiben fast alle Rechte beim Anbieter.					

Tab. 1: Vergleich von Softwaremodellen

Die folgende Grafik[21] verweist auf einen weiteren interessanten Aspekt bei der Einordnung von Open-Source-Software im Vergleich zu den oben genannten Modellen. Open-Source-Software kann als eine Art Schnittstelle zwischen Free Software und der kommerziellen bzw. geschlossenen, nicht freien Software betrachten werden:



**Falls Quellcode vorhanden*

Abb. 1: Schnittstelle „kommerziell – frei"

Der nächste Abschnitt wird zeigen, welche speziellen Open-Source-Projekte unter anderem diese Schnittstelle besetzen.

Erfolgreiche OS-Projekte

Auf Grund der vielen OS-Projekte ist es kaum möglich, eine genaue Angabe über die Anzahl der bis heute entwickelten OS-Software und deren Anwender abzu-

[21] In Anlehnung an (Ploch u.a. 2002, S. 26).

geben[22]. Ein großer Teil der Anwendungen wurde für die Bereiche Internet, Betriebssystem, Emulatoren, Spiele, Multimedia und Entwicklungs- bzw. Programmierwerkzeuge geschrieben. OSS ist in fast allen Einsatzgebieten zu finden, in denen Software genutzt wird – von der Büroanwendung bis zur Bibel-Lern-Software.

Im folgenden Abschnitt soll eine Auswahl von interessanten und erfolgreichen OS-Projekten präsentiert und Software vorgestellt werden, die sich im privaten, öffentlichen wie unternehmerischen Bereich vielseitig bewährt hat.

Das wohl bekannteste Projekt, das immer wieder auch als Zugpferd der OS-Bewegung bezeichnet wird, ist wohl *GNU/Linux*[23]. Es ist ein freies Betriebssystem, das in den letzten Jahren seinen kommerziellen Konkurrenten in vielen Bereichen Marktanteile abringen konnte und das eine hohe Anzahl an freien Softwarepaketen wie z. B. Emacs, Bash, GNU-Compiler-Sammlung, beinhaltet.

Der Name wird oft auf „Linux" reduziert, was von der FSF-Gemeinde scharf kritisiert wird. Wenn von der Migration auf Linux die Rede ist, ist die Umstellung des Betriebssystems auf GNU(-Software) mit den Linux-Kernel gemeint.

Openoffice ist eine umfangreiche Büroanwendung, die Textverarbeitung, das Erstellen von Präsentationen, Zeichnungen und HTML-Dokumenten, Tabellenkalkulation und die Arbeit mit Datenbanken ermöglicht. Sie ähnelt damit bewusst dem Office-Paket von Microsoft. Etwa 16 Millionen Anwender weltweit nutzen inzwischen diese bislang in 30 Sprachen übersetzte Büro-Software, sie kann auf allen gängigen Betriebssystemen installiert werden. Als kleines Beispiel seiner Funktion mag dieses Buch dienen, das fast ausschließlich mit Openoffice gestaltet wurde.[24]

Nachdem Netscape den Quellcode seines Browsers freigegeben hat, entstand in einer riesigen virtuellen Gemeinschaft dessen Weiterentwicklung *Mozilla*[25]. Dieses Programmpaket kann zum Surfen im Internet, zum Schreiben und Lesen von E-Mails und News, zum Chatten, zur Erstellung von Webseiten (per WYSIWYG[26]-Editor) und sogar zum Verwalten von Adressen und Terminen verwendet werden. Der Browser wurde vom PC-World-Magazin mit dem Titel „Best of 2003, Web Browser" ausgezeichnet.

KDE (K Desktop Environment)[27] ist eine grafische Benutzeroberfläche und Arbeitsumgebung für UNIX-PCs (also auch für GNU/Linux). Es beinhaltet eine umfangreiche Sammlung an Anwendungen inklusive eines eigenen Office-Pakets, eines Webbrowsers und einer E-Mail-Anwendung. Das Projekt wurde 1996 in Deutschland gegründet – heute ist KDE die meistgenutzte grafische Benutzeroberfläche auf entsprechenden Desktopsystemen.

---

22 Eine Vielfalt von Projekten findet sich unter anderem auf den folgenden Webseiten: http://freshmeat.net, http://sourceforge.net, http://OSDir.com, http://developer.berlios.de, http://bioinformatics.org.

23 Vgl. http:/www.gnu.org bzw. http://www.linux.org.

24 Vgl. http://www.openoffice.org.

25 Vgl. http://www.mozilla.org.

26 *what you see is what you get*

27 Vgl. http://www.kde.de.

GNOME (GNU Network Object Model Environment)[28] stellt ebenfalls eine grafische Desktop-Umgebung dar. Das Projekt wurde 1997 als Konkurrent[29] zum KDE-Projekt gegründet. Es hat allerdings eine nicht so große Nutzer- und Entwicklergemeinde wie das KDE-Projekt.

Die nun folgenden Software-Projekte werden dem Endanwender vielleicht weniger geläufig sein, für Spezialisten sind sie aber heute nicht mehr wegzudenken.

Samba[30] ist ein File- und Druckserver. Er macht es möglich, auf andere im Netzwerk befindliche Computer (auch auf Windows-Rechner) zuzugreifen, um Dateien auszutauschen oder Druckaufträge abzuschicken.

Den heute verbreitetsten Webserver stellt *Apache*[31]. Ein Webserver ermöglicht es, Seiten der unterschiedlichsten Art im Internet zu veröffentlichen und anzuzeigen. Dieser Server ist ein Projekt der *Apache Software Foundation*, welche noch viele weitere erfolgreiche OS-Projekte betreut.

Zope und *OpenCms*[32] sind zwei sehr erfolgreiche Content-Management-Systeme. Sie erleichtern die Verwaltung von Webinhalten (Content, wie z.B. Texte und Bilder für bestimmte Bereiche wie News oder Unternehmens- bzw. Produktinformationen) von entsprechenden Webseiten. Zope basiert auf der Programmiersprache Python und OpenCms auf Java.

Eines der heute erfolgreichsten Datenbanksysteme ist *MySQL*[33]. Neben der Möglichkeit, diese Software unter der freien Lizenz GPL zu nutzen, kann auch, wenn beispielsweise ein kommerzielles Produkt auf Basis des MySQL-Quellcodes entwickelt werden soll, eine entsprechende kommerzielle Lizenz erworben werden.

Die folgenden beiden OS-Projekte haben die Entwicklung des Internets maßgeblich geprägt: So ist *BIND*[34] eine Software, die das Domain-Name-System (DNS) unterstützt, welches dazu dient, die vom Menschen lesbaren Netz-Adressen (wie z.B. www.opensource.org) in eine für den Computer nutzbare IP-Adresse (bestehend aus Zahlen) aufzulösen.

Sendmail [35] ist ein so genannter *Mail Transport Agent* (MTA). Er dient zur Weiterleitung von E-Mails. Der Marktanteil dieses bereits 1981 entwickelten Programms lag im Jahre 1999 bei ca. 75 %.

---

28 Vgl. http://www.gnome.org.

29 Vgl. auch den Artikel von Eva Brucherseifer: „Die KDE-Entwicklergemeinde – wer ist das?“, Abschnitt: „3. KDE und der Rest der Welt“ in diesem Band.

30 Vgl. http://samba.org, vgl. auch die interessante Nutzerstudie, die unter http://samba.anu.edu.au/pub/samba/survey einsehbar ist.

31 Vgl. http://www.apache.org.

32 Vgl. http://www.zope.org und http://www.opencms.org.

33 Vgl. http://www.mysql.com.

34 Vgl. http://www.isc.org.

35 Vgl. http://www.sendmail.org.

Migrationen, Entwicklergemeinschaft und Buch-Rezension: Die Artikel in diesem Kapitel

In den nächsten Abschnitten des ersten Kapitels sollen Erfahrungsberichte über die Migration von Microsoft-Produkten zu Open-Source-Software, ein Bericht über die KDE-Gemeinschaft und eine Rezension über Steven Webers demnächst erscheinende Darstellung „Success of Open Source" vorgestellt werden.

Es wurden drei Migrationsprojekte für das Einleitungskapitel ausgewählt. Davon wurden zwei Projekte vom Bundesamt für Sicherheit in der Informationstechnik (BSI) bei den Bundesbehörden – nach einer Umfrage auf „Migrationsinteresse" – zum Zweck der Evaluation von OSS europaweit ausgeschrieben und finanziert.

Eines dieser Projekte wird in dem Artikel „Migration auf Open-Source-Software beim Institut für Tierzucht der Bundesforschungsanstalt für Landwirtschaft" aus der Sicht des mit der Migration beauftragten Unternehmens – der GONICUS GmbH – dargestellt. Der Autor Alfred Schröder beschreibt dabei nicht nur die Umstände des Projekts und inwieweit es Erfolg hatte. Er geht auch auf die vorgefundenen Benutzertypen und -profile ein und zeigt entsprechend gesammelte Erfahrungen. Weiterhin betont er die Wichtigkeit der Benutzer bei einer solch gravierenden Umstellung der Bildschirmarbeitsplätze.

Ein weiteres Projekt wurde bei der Monopolkommission durchgeführt. Für dieses Jahrbuch konnte hierzu ein Bericht aus Sicht des Dienstleisters („Beispiel einer Migration von Windows 2000 auf Open-Source-Software" von Thomas Sprickmann Kerkerinck), der die entsprechende Open-Source-Software eingeführt hat, gewonnen werden.

Dieser Migrationsbericht ist in ähnlicher Form schon einmal veröffentlicht worden.[36] Da aber gerade das Thema Migration auf Open-Source-Software und der Rahmen des durchgeführten Pilotprojektes bzw. die daraus gesammelten Erfahrungen als wegweisend angesehen werden können, durfte er in diesem Jahrbuch nicht fehlen. Aus den gemachten Darstellungen kann der Leser sich ein Bild vom Umfang und von der Komplexität einer solchen Migration machen. Einige Ausführungen werden dem Laien sehr technisch erscheinen, sie stellen aber die Kernidee der Umstellung dar und zeigen außerdem, was mit OSS alles möglich ist.

Die bei der Umstellung der Monopolkommission gewonnenen Erfahrungen wurden beim Landesrechnungshof Mecklenburg-Vorpommern verwendet. Den Anwenderbericht aus diesem Projekt finden Sie im Artikel von Frank Müller: „Migration der Server- und Desktoplandschaft im Landesrechnungshof Mecklenburg-Vorpommern".

Einen weiteren Bericht aus der Sicht des Benutzers konnte Kerstin Terhoeven beisteuern. Sie betrachtet das Migrationsprojekt in ihrem Aufsatz „Open-Source-Software am Büroarbeitsplatz: Erfahrungen der Endanwender aus der Migration der Geschäftsstelle der Monopolkommission" eher kritisch. Sie schildert diverse im Vergleich zum vorher eingesetzten System als negativ empfundene Eindrücke. Weiterhin beschreibt sie die Vor- und Nachteile des neuen Systems. Es erschien der Redaktion dieses Buches wichtig, auch die vorhandene Kritik an dieser Thematik ab-

[36] S. z. B. http://www.pl-forum.de/berichte/monopolkommission.html.

zubilden, nicht zuletzt, um auch die Herausforderungen, die eine Migration mit sich bringt, darzustellen.

Um dem Leser eine OS-Entwicklergemeinschaft näher zu bringen, wurde die Projektleiterin und stellvertretende Vorsitzende des KDE e.V. Eva Brucherseifer gebeten, als Insiderin dieses Projekt vorzustellen. In dem Artikel „Die KDE-Entwicklergemeinde – wer ist das?“ beschreibt sie die Anfänge, überwundene Probleme und die Erfolge. Weiterhin berichtet Eva Brucherseifer sehr detailliert vom Leben in einer solchen Gemeinschaft – insbesondere von den Treffen, die auf Grund des sonst kaum vorhandenen persönlichen Kontakts sehr wichtig sind.

Beendet wird das erste in die Thematik einführende Kapitel durch den Hinweis auf weiterführende Literatur. Dazu dient die von Hendrik Scheider verfasste Rezension des im April dieses Jahres erscheinenden Buches von Steven Weber: „Success of Open Source“.

Literatur

Kerbusk, Klaus-Peter (2003): *Mauerfall an der Isar*, in: Spiegel 23/2003, online http://www.spiegel.de/spiegel/0,1518,251077,00.html (24.1.2004).

Bundestux(2002): *Wir können uns das gut vorstellen!*, online http://www2.bundestux.de/bundestux_alt/erklaerung.html (24.1.2004).

Bundestux (2003): *München als Vorhut der Informationsgesellschaft*, , online http://www.bundestux.de/themen/inlkom/12914.html (24.1.2004).

Free Software Foundation (2002): *Die Definition Freier Software*, , online http://www.gnu.org/philosophy/free-sw.de.html (24.1.2004).

Grassmuck, Volker (2002), *Freie Software. Zwischen Privat- und Gemeineigentum*, Bundeszentrale für politische Bildung, Bonn 2002.

Levy, Steven (2001): *Hackers, Heroes of the Computer Revolution*, Penguin Books 2001.

Open Source Initiative (2004): *The Open Source Defintion – Version 1.9*, online www.opensource.org/docs/definition.php (24.1.2004).

O'Reilly (1999): *Open Source – kurz und gut*, O'Reilly 1999, online http://www. oreilly.de/german/freebooks/os_tb/ (24.1.2004).

Ploch, Danuta; Stewin, Patrick; Koch, Ramona: *Einführung in Open Source Software*, Technische Universität Berlin – Informatik und Gesellschaft 2002, online http://ig.cs.tu-berlin.de/w2002/ir1/uebref/PlochEtAl-Referat-G1-Referat-122002.pdf (24.1.2004).

Stallman, Richard (2001): *Das GNU Projekt*, Originalveröffentlichung in dem Buch „Open Sources“, übersetzt 2003 von Stephan Knuth, Free Software Foundation 2003, online http://www.gnu.org/gnu/thegnuproject.de.html (24.1.2004).

Migration auf Open-Source-Software beim Institut für Tierzucht der Bundesforschungsanstalt für Landwirtschaft

ALFRED SCHRÖDER

1. Einleitung

Mitte 2002 schrieb das Bundesministerium des Inneren über das Bundesamt für Sicherheit in der Informationstechnik drei Pilot-Migrationsprojekte für unterschiedliche Institute und Behörden aus.

Bei einer dieser Institutionen, dem Institut für Tierzucht der Bundesforschungsanstalt für Landwirtschaft fand sich eine etwas ungewöhnliche Umgebung.

Im Rahmen der Pilotprojekte sollte die Einsatzfähigkeit von GNU/Linux für die öffentliche Verwaltung untersucht werden. Dabei ging es sowohl um den Einsatz von GNU/Linux als Server- als auch als Desktop-System. Die gewonnenen Erfahrungen sollten gesammelt und anderen Behörden, die ebenfalls vor Migrationsentscheidungen stehen, zur Verfügung gestellt werden. Angesichts der Innovationszyklen in der IT und der damit verbundenen „vor der Tür stehenden" Migrationsentscheidungen sollte eine Einschätzung von Alternativen ermöglicht werden, die im Hinblick auf Kriterien wie Sicherheit, Stabilität, Unabhängigkeit und Wirtschaftlichkeit neue Möglichkeiten für den Bereich der öffentlichen Hand eröffnen.

Um es vorwegzunehmen: Sowohl die beteiligten Behörden und Unternehmen als auch insbesondere das BSI waren durchaus positiv überrascht von den Ergebnissen. Auf Basis der gewonnenen Erkenntnisse kann gesagt werden: GNU/Linux ist für den Behördendesktop durchaus geeignet.

Nun stellt – wie bereits oben eingeleitet – das Institut für Tierzucht sicherlich keine exemplarische Behörde dar.

Durch die Verbindung von Wissenschaft und Verwaltung bieten sich hier aber besondere Herausforderungen, die die gemachten Erfahrungen sogar universeller einsetzbar machen.

Entsprechend möchte ich in diesem Bericht auch insbesondere auf die Erfahrungen der unterschiedlichen Benutzertypen und -profile eingehen. Die gemachten Beobachtungen sind dabei sicher durch meine Einbindung als externer Projektleiter gefärbt. Auf Grund der engen Zusammenarbeit mit den Benutzern, die ein solches Desktop-Projekt mit sich bringt, sind aber auch viele interessante Situationen entstanden, deren Schilderung ein gutes Bild der Situation und des Projektes zeichnen kann.

2. Ausgangssituation

Die Zeit war knapp! Von Beginn an war der Zeitrahmen, der für die Umstellung der Umgebung zur Verfügung stand, knapp bemessen. Mit nicht einmal drei Monaten für die Konzeption und Umsetzung einer – wenn auch nur teilweisen – Migration einer EDV-Landschaft war allen Beteiligten klar, dass ein solch sportlicher Zeitrahmen nur durch enge Zusammenarbeit und koordiniertes Vorgehen einzuhalten war.

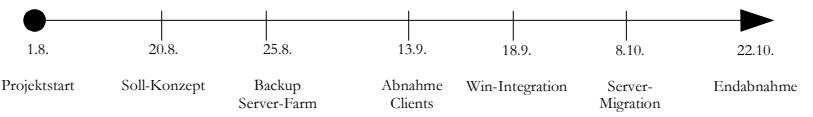


Abbildung 1: Zeitplan der Migration

Wie stellte sich die Situation aber neben dieser alles bestimmenden Prämisse dar? Die IT des Instituts für Tierzucht zeigte sich als eine über lange Zeit und unterschiedliche Betriebssystemgenerationen gewachsene Landschaft. Auf der Serverseite wurden sowohl bereits GNU/Linux als auch Solaris neben Windows eingesetzt. Auf Grund der in diesem Bereich bereits gemachten Erfahrungen galt nun das Hauptaugenmerk der Client-Seite. Hier wurden unterschiedlichste Windows-Versionen (Windows 98–Windows XP) neben Macintosh- und OS/2-Rechnern eingesetzt. An einzelnen Arbeitsplätzen existierten auch schon Linux-Workstations für Entwickler.

Die Aufgaben, die an all diesen Arbeitsplätzen erledigt werden, erstrecken sich über ein weites Spektrum und reichen von den eher klassischen Büro- und Verwaltungstätigkeiten in einer Behörde bis zur Unterstützung wissenschaftlicher Versuche, zur Ansteuerung von Laborgeräten, Programmentwicklung, Bildbearbeitung und Ausarbeitung von wissenschaftlichen Arbeiten.

Im Rahmen des Pilotprojektes sollten nun sowohl die benötigte serverseitige Infrastruktur aufgebaut als auch 30 Bildschirmarbeitsplätze auf GNU/Linux migriert werden. Das heißt aber natürlich neben der technischen Lösung bezüglich der Rechnersysteme auch die Umstellung der Benutzer mit allen Aspekten, die diese Aufgabe umfasst. Um ein möglichst breites Spektrum abzudecken, sollten diese Benutzer und Arbeitsplätze aus den unterschiedlichsten Bereichen der Behörde stammen.

Die wohl wichtigsten Forderungen im Vorfeld des Projektes, die selbstverständlich einen entscheidenden Einfluss auf den Projektverlauf hatten, sollen im Folgenden aufgeführt und beleuchtet werden.

Einbindung der EDV-Mitarbeiter

Die örtlichen Mitarbeiter der EDV des Instituts hatten sich, bestärkt durch ihre teilweise vorhandenen ersten Erfahrungen mit GNU/Linux, bereits im Vorfeld für das Pilotprojekt engagiert. Gerade deshalb und natürlich auch im Hinblick auf einen unabhängigen effektiven Betrieb der Lösung war die vollständige und umfassende Einbindung dieser Mitarbeiter in den Projektablauf eine wichtige Voraussetzung.

Diese Einbindung sollte dabei die Mitarbeiter letztendlich nicht nur in die Lage versetzen, die geschaffene Umgebung effektiv zu verwalten, sondern auch, die mit dem Pilotprojekt begonnene Migration des gesamten Instituts eigenständig vollenden zu können.

Zentrale Administration

Wegen der Erfahrungen, die man im Institut mit der Verwaltung einer großen Anzahl von Desktop-Systemen sammeln konnte, und der intensiven Vorüberlegungen, die angestellt wurden, hatte man sich auch schon auf einen weiteren wichtigen Grundsatz festgelegt: Die neu zu konzeptionierende Umgebung sollte einen Fokus auf eine zentrale und einfache Verwaltung legen. Aus diesem Grund wurde bereits im Vorfeld der Gedanke einer ThinClient[1]-Umgebung favorisiert.

Unterstützung lokaler Peripherie?

Auf Grund des wissenschaftlichen Charakters vieler Arbeitsplätze und obwohl man einen ThinClient-Ansatz favorisierte, war es wichtig sicherzustellen, dass lokal angeschlossene Peripherie an den Clients unterstützt wird. Nur so wäre möglich, dass diverse wissenschaftliche Geräte eingesetzt und somit die entsprechenden Arbeitsplätze in die Migration mit einbezogen werden können.

3. Das Projekt

3.1. Vorbemerkungen

GNU/Linux gilt als etabliertes System – auf der Serverseite. Dort ist der Einsatz für den Endanwender transparent und bedeutet keine sichtbare Veränderung. Auf dem Desktop hingegen sieht das naturgemäß etwas anders aus. Hier ist der entscheidende Faktor der Benutzer. Auf ihn hat die Veränderung der Arbeitsumgebung unmittelbare Auswirkungen. Unzufriedene Anwender können ein Projekt schnell zum Kippen bringen, da eine Migration gegen den Willen der Endanwender nicht oder nur sehr schwer möglich ist. Auf der anderen Seite können motivierte Mitarbeiter, die den Schritt hin zu GNU/Linux mittragen, erheblich zu einem positiven Projektverlauf beitragen. Entsprechend galt von Beginn des Projektes an der Einbeziehung der Anwender eine erhöhte Aufmerksamkeit.

In den ersten Projekttagen war deutlich geworden, dass die in Betracht kommenden Anwender bis dahin nicht oder nur unzureichend über die anstehenden Schritte informiert waren. Angesichts des Projektstarts hatten sich dann aber bereits die unterschiedlichsten Gerüchte gebildet, und es waren Befürchtungen in die eine oder andere Richtung aufgebaut worden. Um dieser Entwicklung entgegenzuwirken, sind kurzfristig Informationsveranstaltungen angesetzt worden.

Im Rahmen dieser Veranstaltungen, in denen unterstützt durch die Institutsleitung die Motivation und die Ziele des Migrationsprojektes erläutert wurden, ist den Mitarbeitern auch ein erster Eindruck eines Linux-Desktops vermittelt worden. Es wurde versucht, Berührungsängste abzubauen und durch die Beantwortung von Fragen dem Wildwuchs an Befürchtungen entgegenzuwirken.

[1] Schlanke Rechnersysteme ohne lokalen Massenspeicher, die über das Netzwerk ihr Betriebssystem beziehen und völlig zentralistisch zu verwalten sind.

Als Beispiel für die Befürchtungen kann sicherlich die folgende Frage dienen, die aufkam, nachdem bekannt gegeben wurde, dass eine ThinClient-Infrastruktur angestrebt wird: „Nehmen Sie uns dann die Computer weg und wir müssen auf dem Netz arbeiten?“ Diese auf den ersten Blick amüsante Frage steht natürlich auch symptomatisch für die Befürchtung der Anwender, die Kontrolle über das eigene System und dessen Status zu verlieren.

Die offene Auseinandersetzung mit diesen Befürchtungen im Rahmen einer Informationsveranstaltung war aber zumindest ein erster Baustein, die Benutzer für das Projekt zu gewinnen.

3.2. Analysephase

Auch wenn der Zeitdruck zur Durchführung des Projektes groß war, so bestand kein Zweifel, dass auch oder insbesondere in solch einer Situation größte Sorgfalt anzuwenden ist. Das Projekt startete also mit einer umfassenden Bestandsaufnahme im Hinblick auf die unterschiedlichen Aspekte der bestehenden EDV-Umgebung. Dabei wurden sowohl technische Fragen wie Rechnerausstattung oder Applikationen als auch die unterschiedlichen Benutzertypen hinsichtlich ihrer Erfahrungen und Vorbildung und nicht zuletzt organisatorische Fragen betrachtet.

Es wurde eine möglichst genaue Einteilung der zu migrierenden Benutzer in Lerngruppen auf Basis von umfangreichen Interviews vorgenommen. Auf diese Weise sollte für die anstehenden Schulungen sichergestellt werden, dass weitestgehend homogene Gruppen zusammen unterrichtet werden.

Ebenso wurden die Erwartungen und Anforderungen der am Projekt beteiligten Mitarbeiter aufgenommen, um die Berücksichtigung dieser Ideen zu gewährleisten. Durch die Einbindung dieser Ideen und Vorschläge kann die Identifizierung mit dem Projekt weiter gefördert werden.

3.3. Konzeption

Auf Basis der in der Analysephase gewonnenen Erkenntnisse, der Erfahrungen der Berater im Projekt und eines kontinuierlichen Ideenaustauschs mit den EDV-Mitarbeitern vor Ort (Feedback) entstand ein umfassendes Konzept, welches den Rahmenbedingungen, den Anforderungen, den Benutzern, dem Zeitrahmen und den technischen Möglichkeiten Rechnung trug.

Als Ergebnis der angestellten Überlegung hatten sich alle Projektbeteiligten auf eine Struktur geeinigt, wie man sie der folgenden Skizze entnehmen kann.

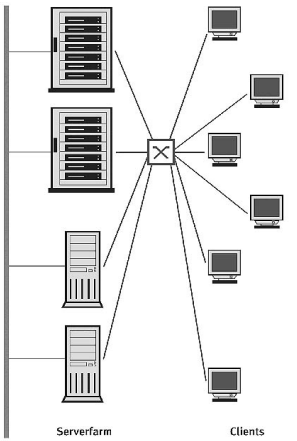


Abbildung 2: Infrastruktur

Basis der neu zu schaffenden Infrastruktur bildete die Serverfarm[2], die das Anwendungs- und Dienstangebot für die Benutzer bereitstellt. Diese Farm gliedert sich in die zentralen Fileserver[3], mehrere Linux-Applikationsserver[4] und zwei Citrix-Server[5], die durch einige Server mit Spezialaufgaben komplettiert wurden.

Das Rückgrat der Konzeption bildete ein zentraler Verzeichnisdienst, der alle relevanten Informationen bezüglich der Benutzer, der Applikationen und der Systeme beinhaltet und somit auch den Punkt für die zentrale Administration bildet.

Auf der anderen Seite stehen die ThinClient-Systeme. Hier kam im Zusammenspiel mit der Serverfarm das an das Linux-Terminal-Server-Projekt angelehnte GOto[6] zum Einsatz, bei dem die Clients vollständig über das Netz booten. Um dies zu ermöglichen, werden PXE-fähige[7] Netzwerkkarten eingesetzt. Diese Karten senden beim Start des Rechners eine Anfrage ins Netz, die von den zentralen File-Servern mit einem Basis-Linux-System beantwortet wird. Dieses Basissystem wird sodann dem Client zur Verfügung gestellt, sodass dieser seinen Boot-Vorgang starten kann.

Im Verlauf des Boot-Vorgangs überprüft der Client, ob er selbst schon im zentralen Verzeichnisdienst aufgeführt ist. Sollte das nicht der Fall sein, wird eine Hardwareerkennung gestartet, um die vorhandenen Komponenten und die zugehörige GNU/Linux-Unterstützung zu identifizieren. Die Ergebnisse dieser Erkennung werden dann in den Verzeichnisdienst geschrieben. Da der Client in diesem Fall

---

2 Eine Ansammlung von Servern, die gemeinsam Dienste für den Anwender im Netzwerk zur Verfügung stellen.

3 Server zur Ablage von Dateien und Dokumenten der Benutzer.

4 Server, auf denen die Anwender Applikationen starten können. Ein- und Ausgaben dieser Anwendungen erfolgen dann über die ThinClients, sodass der Benutzer die Ergebnisse an seinem Arbeitsplatz vorliegen hat.

5 Spezielle Applikationsserver für Windows-Anwendungen.

6 Siehe http://www.GONICUS.de.

7 PXE ist eine Spezifikation, die es Rechnern ermöglicht, ohne lokalen Massenspeicher direkt über ein angeschlossenes Netzwerk ein Betriebssystem zu laden, um so den Rechner zu starten.

aber als ‚dem System unbekannt' gelten muss, wird der Boot-Vorgang erst dann fortgesetzt, wenn der Systemadministrator eine Freigabe über den Verzeichnisdienst erteilt. Falls der Client sich selbst aber bereits im LDAP[8]-Baum wiedergefunden hat, wird der Bootvorgang direkt und schnell anhand der dort abgelegten Konfigurationsinformationen durchgeführt.

Nach diesem Schritt überprüft der ThinClient die Verfügbarkeit der Linux-Applikationsserver und sucht sich denjenigen mit der geringsten Last. Mit diesem verbindet sich der Client. So wird dem Benutzer eine Anmeldung auf dem Server zur Verfügung gestellt, auf dem auch seine Anwendungen ausgeführt werden.

Alle nativ unter GNU/Linux verfügbaren Anwendungen stehen somit unmittelbar zur Verfügung. In diesem Szenario wurden damit die Bereiche Office-Suite (Openoffice), Internet-Zugriff (Mozilla) und E-Mail (KMail) direkt abgedeckt. Diese Umgebung wird durch den Einsatz weiterer unter GNU/Linux verfügbarer Applikationen auf einem angepassten KDE-Desktop abgerundet, der gemeinsam durch Berater, Benutzer und Administratoren konfiguriert wurde.

Die über dieses Spektrum hinaus benötigten Anwendungen, die nicht direkt unter GNU/Linux zur Verfügung standen, sollten möglichst nahtlos und transparent in die Umgebung eingebunden werden. Auf Grund des Pilotcharakters wurden dabei unterschiedliche Möglichkeiten angewandt, um ein breites Spektrum an Erfahrungen sammeln zu können. So kamen im Rahmen dieses Pilotprojektes sowohl Citrix-Terminal-Dienste und VMWare (Rechner-/Windows-Emulator) als auch DOSEmu (DOS-Emulator) für unterschiedliche Aufgaben zum Einsatz.

Um die wichtige Anforderung nach einer einfachen und zentralen Administration zu erfüllen, bildete der Einsatz des zentralen Verzeichnisdienstes die Grundlage. Um die Administration aber auch komfortabel bewerkstelligen zu können, wurde hier das freie LDAP-Web-Front-End GOsa[9] eingesetzt, welches für die in dieser ThinClient-Umgebung anfallenden tagtäglichen Aufgaben speziell gerüstet ist. Zusammen mit einer Reihe von Skripten zur Unterstützung von Aufgaben, die sich insbesondere mit der Skalierung der Umgebung wie dem Klonen von Applikationsservern befassen, stand somit am Ende dieser Phase ein gemeinsam erarbeitetes Konzept, das von allen Seiten getragen werden konnte.

3.4. Die Umsetzung

Da für das Projekt auf zum Teil bereits produktiv im Einsatz befindliche Hardware zurückgegriffen werden musste, stellte GONICUS zuerst einmal eine Ausweichserverfarm zur Verfügung, auf der teilweise die Ergebnisse der konzeptionellen Phase umgesetzt wurden. Auf diese Weise waren ein paralleler Betrieb von zwei Welten und ein ausführlicher Test mit unterschiedlicher Client-Hardware möglich.

Zuerst wurde also die Serverfarm gemäß den aufgestellten Konzepten realisiert, um an dieser Serverfarm dann die ersten Client-Systeme testen zu können. Im Vorfeld dieser Umsetzung – bereits während der Konzeption – wurden die Schulungsmaßnahmen für die Administratoren begonnen. Auf diese Weise sollte die Möglich-

[8] LDAP: Lightweigt Directory Access Protocol – bietet einen Verzeichnisdienst, in dem beliebige Informationen in Objekten gespeichert werden können. s. auch http://verzeichnisdienst.de/ldap/.

[9] Siehe https://gosa.GONICUS.de.

keit geschaffen werden, die Administratoren frühzeitig aktiv in die Umsetzung einzubinden und so von Beginn an eine hohe Identifizierung mit der Umgebung zu gewährleisten.

Gemeinsam mit ausgewählten Pilotbenutzern konnten dann erste Client-Systeme für die neue Umgebung getestet werden. Hier war die Einbeziehung von aufgeschlossenen Benutzern gefragt, die durch den Einsatz in ihrem Büroalltag wertvolle Anregungen zu Verbesserungen und Umgestaltungen liefern konnten. Um einen möglichst umfassenden Eindruck zu gewährleisten, waren dabei auch Benutzer aus den unterschiedlichsten Fachbereichen vorgesehen.

Das deutliche Interesse an der neuen Umgebung manifestierte sich dann aber auch schnell in einem umfassenden Interesse der Kolleginnen, die die Aufgabe eines Test-Users übernommen hatten. Es stellten sich in diesem Zusammenhang auch schnell Aha-Effekte ein.

So konnten wir an einem späten Nachmittag einmal mehr eine Traube von Mitarbeitern um den Arbeitsplatz eines Test-Benutzers sehen, die voller Begeisterung gemeinsam KTetris[10] spielte – voller Verwunderung, dass es doch sogar auch Spiele unter KDE gibt. Die Freiheit, auch solch „unproduktive Elemente" in die Testumgebung zu integrieren, war auf expliziten Wunsch der Instituts-IT und als kleiner Anreiz zur Auseinandersetzung mit der Umgebung vorgesehen worden.

Nach der Einarbeitung der Anregungen durch die ausgewählten Benutzer und Administratoren stand die eigentliche Migration der gesamten Pilotteilnehmer an. Alle Pilotteilnehmer erhielten eine an die in ihren Bereichen auftretenden Anforderungen angepasste Schulung im Umfang von weniger als einer Woche, die sowohl Grundlagen der neuen Oberfläche als auch die neue Office-Suite adressierten. Auf Grund der besonderen Anforderungen des Instituts gab es aber auch noch spezielle Schulungen in den zwei Themenbereichen „Bildbearbeitung mit Gimp?" und „Erstellung wissenschaftlicher Arbeiten auf LaTex-Basis mit dem Front-End Lyx".

Selbstverständlich konnten nicht alle Benutzer zeitgleich geschult werden, sodass passende Gruppen gebildet wurden. Gemeinsam mit den örtlichen EDV-Mitarbeitern konnte sichergestellt werden, dass die Mitarbeiter-Arbeitsplätze und die Dateien der Benutzer dann umgestellt bzw. transferiert wurden, wenn diese Mitarbeiter gerade in der Schulung waren. Die Schulungen wurden dabei auf Systemen durchgeführt, die genau den später am Arbeitsplatz anzutreffenden Umgebungen entsprachen. Auf diese Weise konnte gesichert werden, dass die durch die Schulung an die Umgebung gewöhnten Benutzer an ihrem Arbeitsplatz direkt und nahtlos in der neuen Umgebung weiterarbeiten konnten.

Natürlich wurde auch sichergestellt, dass die Benutzer bei der Rückkehr an ihren Arbeitsplatz nochmals eine kurze Einweisung erhielten und dass Ansprechpartner (externen Berater gemeinsam mit den EDV-Mitarbeitern des Instituts) für die eventuell auftauchenden Fragen bereitstanden.

Eine besondere Herausforderung bildeten bei diesen Client-Umstellungen die Arbeitsplätze mit ungewöhnlicher Peripherie. Im Rahmen des Projekts wurden da-

[10] Eine Tetris-Variante.

bei so exotische Gerätschaften wie Pipettierroboter oder Gensequenzer, aber auch einfache Digitalkameras oder Videoserver eingebunden.

Den letzten Part bei der Umsetzung bildete dann die Übertragung der neuen Umgebung von der Backup-Server-Farm auf die produktiven Systeme, die dabei völlig überschrieben wurden. Da neben den Pilot-Benutzern einige dieser Server aber auch von allen übrigen Anwendern als Fileserver benutzt wurden, musste an dieser Stelle besondere Sorgfalt darauf verwandt werden, einen nahtlosen Übergang zu gewährleisten. Umso mehr, als dass in diesem Zuge für alle Benutzer die Authentifizierung gegen den Verzeichnisdienst eingeführt wurde. Die Arbeiten mussten also in einem Service-Fenster durchgeführt werden, in dem die Instituts-EDV den Anwendern nicht zur Verfügung stehen konnte.

Mit dieser Umstellung konnte die Pilotumgebung dann in den Regelbetrieb übernommen werden. An dessen Anfang natürlich – auch trotz aller Tests – individuelle Anpassungen an besondere Benutzer- bzw. Arbeitsplatzanforderungen standen.

4. Benutzer-Erfahrungen

Im Rahmen der Konzeption, der Umsetzung, aber auch des Supports nach der Umsetzung wurden die Berater vor Ort mit den unterschiedlichsten Benutzertypen und -anforderungen konfrontiert. Die enge Zusammenarbeit mit den EDV-Betreuern des Instituts, die über eine optimale Einschätzung ihrer Benutzer verfügten, war bei der Bewältigung dieser Herausforderungen von großer Bedeutung.

Das Spektrum der Reaktionen und der Einstellung der Benutzer gegenüber dem Projekt hatte dabei eine große Bandbreite und veränderte sich im Laufe des Projekts auch erheblich auf Basis neuer Erkenntnisse und Eindrücke. So wich an vielen Stellen anfängliche Unsicherheit und Skepsis einer interessierten Beschäftigung mit den neuen Möglichkeiten.

Die beiden extremsten Pole fanden sich einerseits in dem experimentierfreudigen Anwender, der schon einmal ein Linux ausprobiert hatte, und andererseits im Windows-Profi, der im Allgemeinen zuerst einmal allem aus der Linux-Welt gegenüber ablehnend eingestellt ist. Zwischen diesen Polen fand sich eine Vielzahl unterschiedlicher Schattierungen, die es für den Erfolg des Projektes einzufangen galt.

Auffällig war aus unserer Sicht die positive Resonanz auf die angepassten Schulungen und die Betreuung der Anwender. Hier wurde deutlich, dass die meisten Anwender selten Schulungen insbesondere in Bezug auf ihre Textverarbeitung oder Tabellenkalkulation genossen hatten. Die Folge war ein ineffektiver Einsatz dieser alltäglichen Hilfsmittel, der zudem beim Dokumentenaustausch auch noch zu Problemen mit der Kompatibilität bezüglich Layouts führen konnte. Hier wurde klar, dass der durchschnittliche Benutzer nur mit einem Bruchteil seiner Büroanwendungen wirklich arbeitet und einen noch kleineren Teil umfassend beherrscht. Durch die Schulung des passenden Basiswissens bei einem Openoffice konnten viele Benutzer neben den Spezialitäten eines Openoffice eben auch Grundlagen-Wissen erwerben, das ihnen die tägliche Arbeit erleichtert. Entsprechend offener gingen diese Benutzer dann mit der Umgebung um.

Schwieriger wurde es hingegen mit den Anwendern, die keine Zeit für diese Schulungen fanden und sich auf Basis ihres Wissens mit der neuen Umgebung auseinander setzen mussten oder wollten. Diese Nutzer wurden damit konfrontiert, dass bestimmte Work-Arrounds, die sie sich angeeignet hatten, so nicht mehr funktionieren, und hatten entsprechend wesentlich größere Probleme, sich auf die neue Umgebung einzulassen. Interessanterweise betraf dies in erster Linie Institutsmitarbeiter, die in der Hierarchie eine höhere Stellung einnehmen. Gerade im Sekretariatsbereich der Verwaltung hat man sich hingegen sehr interessiert und offen mit dem Linux-Desktop auseinander gesetzt. Das Interesse ging so weit, dass einige Mitarbeiterinnen des Instituts Mitarbeitern aus dem Migrationsteam nach einiger Zeit Kniffe im KDE zeigen konnten, die diese bis dahin selbst noch nicht kannten.

Die Benutzer in den klassischen Verwaltungsbereichen zeigten entgegen unseren Erwartungen teilweise sogar die größere Bereitschaft sich, auf das neue System einzulassen, als die im wissenschaftlichen Bereich. Dies mag natürlich auch an den vielfältigen besonderen Anforderungen liegen, die im wissenschaftlichen Bereich für einen größeren Umstellungsaufwand sorgten als bei der überschaubaren Anwendungslandschaft im Verwaltungsbereich.

Wirklich erstaunlich für uns war aber die Begeisterung für Lyx und LaTex zur Erstellung wissenschaftlicher Arbeiten, da es aus unserem Verständnis heraus äußerst weit von den bis dahin zu findenden Methoden der Arbeitserstellung entfernt lag. Die umfangreichen Möglichkeiten und der Komfort, den dieses Gespann aber gerade für die Aufarbeitung wissenschaftlicher Texte geboten hat, überzeugte in Verbindung mit der Schulung viele Anwender, einen neuen Weg einzuschlagen.

Insgesamt überwogen an den meisten Stellen Experimentierfreude und Begeisterung für eine neue EDV-Umgebung deutlich über die teilweise sicher auch anzutreffende Skepsis. Getrieben von dieser Begeisterung, fanden sich im Laufe des Projekts auch viel mehr interessierte Anwender, die gerne an den Tests der neuen Umgebung teilgenommen hätten, als es offiziell im Projekt vorgesehen war.

Diese Begeisterung galt insbesondere auch für die Administratoren vor Ort, die aktiv an den Umstellungen mitgewirkt haben. So entstand hier schnell der Beschluss, weitere Arbeitsplätze umzustellen und sich nicht auf die vorgesehenen dreißig Plätze zu beschränken. Durch die Integration und den ständigen Know-how-Transfer war man dann auch in der Lage, schnell selbstständig weitere Umstellungen vorzunehmen, sodass wir sogar gezwungen waren, diesen Eifer zu bremsen. Aus unserer Sicht konnte bei einer so schnellen Ausdehnung der Umstellungen nämlich nicht mehr die ausreichende Schulung und Betreuung der Benutzer gewährleistet werden, die sich im Verlauf des Projekts als überaus wichtig erwiesen hatten. Doch auf Grund der knappen Zeit musste man sich aus unserer Sicht auf die Bereinigung der kleinen Problemchen konzentrieren, bevor man weitere Umstellungen anging.

Trotz unseres Bemühens, die Migrationen langsamer fortzuführen, waren am Ende aber nicht nur 30 umgestellte Arbeitsplätze, sondern mehr als 40 Rechner und noch mehr Benutzer zu verbuchen, die mit GNU/Linux auf dem Desktop arbeiteten. Diese Umstellungen sind nach dem Ende des eigentlichen Projektes dann auch ohne Unterstützung durch GONICUS weiter aktiv fortgeführt worden.

Im Laufe des Einsatzes an allen Pilotarbeitsplätzen wurden dann aber auch Herausforderungen deutlich, die häufig eher organisatorischer als technischer Natur sind. Als Beispiel sei hier der Dokumentenaustausch mit anderen Behörden genannt. Obwohl es offizielle Empfehlungen gibt, die für den Austausch von Layoutgetreu darzustellenden Dokumenten das pdf-Format vorsehen, wird in der Praxis anders verfahren und trotzdem auf die proprietären Microsoft-Formate zurückgegriffen. Dadurch entstanden zusätzliche Anforderungen an die Konfiguration der Standarddateiformate.

5. Fazit

Wie bei Pilotprojekten, in denen es ja ein erklärtes Ziel ist, Erfahrungen von allen Beteiligten zu sammeln, lief auch das Projekt der OSS-Migration des Instituts für Tierzucht sicherlich nicht völlig reibungslos, und es gab an verschiedenen Stellen Herausforderungen, die es gemeinsam mit allen Beteiligten zu bewältigen galt. Rückblickend kann hier aber gesagt werden, dass mit diesem Projekt einmal mehr nachgewiesen werden konnte, dass GNU/Linux für den Einsatz auf dem Desktop geeignet ist. Hier wurde aber zusätzlich gezeigt, dass es speziell auch für die Anforderungen beim Einsatz in Behörden Vorzüge ausspielt, die eine genaue Betrachtung auch bei anderen öffentlichen Einrichtungen lohnend erscheinen lassen.

Die Anwender müssen bei einer Migration, die den Desktop einbezieht, als zentraler Faktor verstanden und entsprechend berücksichtigt werden. Das so ausgerichtete Vorgehen mit passenden, individuellen Schulungen, einer guten Betreuung und der Einbeziehung der Endnutzer hat sich innerhalb des Projektes als überaus positiv erwiesen. Durch eine aktive Informationspolitik konnten viele Ängste, die bei den Anwendern im Vorfeld einer solch umfassenden Umstellung zwangsläufig vorhanden sind, ausgeräumt werden. Trotzdem blieb die Auseinandersetzung mit erklärten Migrationsgegnern auch während des gesamten Projekts eine wichtige Aufgabe, mit der sich die Berater vor Ort beschäftigen mussten. Gerade deshalb hat sich gezeigt, dass es bei den Projektbeteiligten neben der technischen auch insbesondere auf die soziale Vorgehens-Kompetenz ankommt.

Die enge Einbindung der Instituts-EDV hat für den Migrationsprozess einen wertvollen Beitrag geleistet. Da diese Mitarbeiter die Umgebung betreiben müssen, ist ihr Verständnis von der Umgebung entscheidend für einen andauernden und fortgesetzten Erfolg. Auf der anderen Seite ist aber auch deutlich geworden, dass bei dem Einsatz einer vollständig neuen Technologie in einem so kurzen Zeitraum zwar das Basiswissen vermittelt, aber keine umfassende Erfahrung aufgebaut werden kann. Bei Migrationen mit einem solch engen Zeitplan empfiehlt es sich deshalb, im Nachgang zum eigentlichen Projekt noch für einige Zeit eine Zugriffsmöglichkeit auf einen professionellen und erfahrenen 2nd und 3rd Level Support zu haben. Die Vereinbarung von passenden Service-Level-Agreements stellt die Verfügbarkeit von passender Unterstützung sicher, die bei grundlegender Technologie, von der die Funktionalität der Gesamtlandschaft abhängt, anzuraten ist.

Angesichts eines engen Zeitplans, eines extrem heterogenen Umfelds und schwierigen technischen Rahmenbedingungen konnte von allen Seiten, dem Institut

für Tierzucht, dem Bundesamt für Sicherheit in der Informationstechnik und den beauftragten Unternehmen, ein positives Fazit gezogen werden.

Dieses positive Fazit schlägt sich auch in der eigenständigen Fortsetzung des Migrationsprojekts nieder. So haben die Mitarbeiter der Instituts-EDV die Umstellung der Arbeitsplätze weiter fortgesetzt und verfolgen das Ziel der vollständigen Umstellung der gesamten EDV-Landschaft.

Über die GONICUS GmbH:

Die GONICUS GmbH ist ein unabhängiger Open-Source-Dienstleister, der im Februar 2001 von erfahrenen OSS-Spezialisten gegründet wurde. Die rund 15 Mitarbeiter aus den verschiedenen Bereichen (Consulting, System Engineering, Sales) verfügen über langjähriges und umfassendes Know-how im Open-Source-Umfeld und dort speziell beim Einsatz von GNU/Linux. Der Hauptsitz von GONICUS befindet sich in Arnsberg bei Dortmund. Die deutschlandweite Beratungs- und Projekttätigkeit wird durch eine Repräsentanz in Bonn erleichtert. Die Tätigkeiten der GONICUS in den Bereichen Consulting, Implementation, Support und Training fokussieren auf die Bereiche Hochverfügbarkeit (Cluster), ThinClient-Konzepte und die betriebswirtschaftliche Betrachtung des Einsatzes von OSS.

Beispiel einer Migration von Windows 2000 auf Open-Source-Software

THOMAS SPRICKMANN KERKERINCK

1. Die Ausschreibung des BSI

Im Juni 2002 wurde seitens des Bundesinnenministers die Initiative ergriffen Open-Source-Projekte in Bundesbehörden zu initiieren, um die Einsatzfähigkeit von Open-Source-Software (OSS) und im speziellen Linux zu evaluieren. Das Bundesamt für Sicherheit in der Informationstechnik (BSI) suchte daraufhin Behörden, die im Rahmen eines Pilotprojektes bereit waren, eine entsprechende Umstellung durchzuführen, seitens der Monopolkommission war man dazu bereit. Die Ausschreibung wurde mit einer Bieterkonferenz durchgeführt, in der alle namhaften Unternehmen aus der Open-Source-Software-Szene vertreten waren. Am 31. Juli wurde der natural computing GmbH, Dortmund, und ihren Partnern, der SFI Technology Service AG, Schweiz, sowie der Quelltext AG, ebenfalls Dortmund, der Zuschlag erteilt, und am 1. August begann das Projekt.

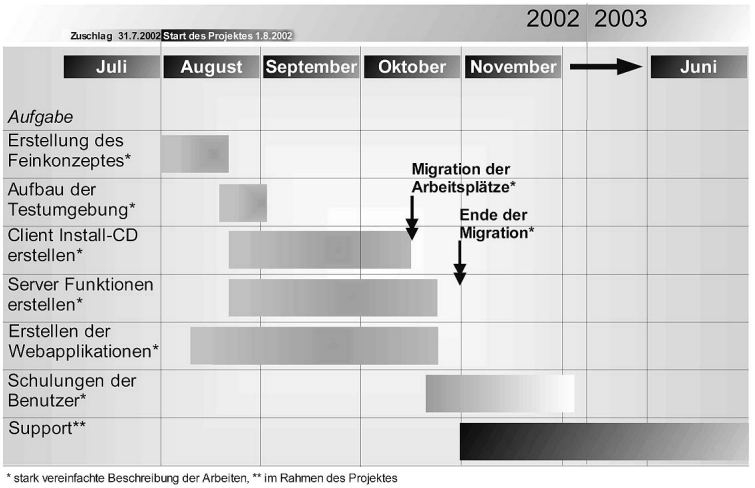


Abbildung 1: Zeitplan des Projektes der Migration der Monopolkommission

1.1. Die Aufgaben der Monopolkommission als Determinante für die IT-Struktur

Um im Rahmen des Projektes eine gute Leistung abgeben zu können, war es uns wichtig zu verstehen, welche Aufgaben die Monopolkommission hat. In der Monopolkommission arbeiten Wissenschaftler als Gutachter für die Bundesregierung an

der Erstellung von Gutachten zur regelmäßigen Beurteilung des jeweiligen Standes und der absehbaren Entwicklung der Unternehmenskonzentration in Deutschland unter wirtschafts-, insbesondere wettbewerbspolitischen Gesichtspunkten, der Würdigung der Vorschriften über die Zusammenschlusskontrolle und der Stellungnahme zu aktuellen wettbewerbspolitischen Fragen. Die Kommission besitzt gegenüber Unternehmen kein Auskunftsrecht. Amtliche Daten unterliegen den jeweils für sie geltenden Vertraulichkeits- und Geheimhaltungsvorschriften.

Im Zusammenhang mit diesen Aufgaben nutzen die wissenschaftlichen Mitarbeiter den Arbeitsplatzrechner, um Texte zu erstellen, Daten auszuwerten und zu präsentieren und sie mit anderen per E-Mail auszutauschen. Die Daten werden aus unterschiedlichen Bereichen und in unterschiedlichen Formaten zur Verfügung gestellt oder über Internetrecherche erhoben. Neben den wissenschaftlichen Mitarbeitern und ihren Aufgaben hat die Monopolkommission weitere Mitarbeiter, die unterschiedliche Aufgaben wie Bibliotheksverwaltung und Sekretariat und die Aufgaben der Geschäftsstelle der Monopolkommission übernehmen. Auch diesen Mitarbeitern sind in der Hauptfunktion Textverarbeitung und Tabellenkalkulation sowie E-Mail zuzuordnen. Darüber hinaus sind jedoch auch die Zusammenstellung von Zeitungsquellen, CD-Archivierung, allgemeine Funktionen wie Terminkalender, scannen, Webseiten erstellen, Handouts erstellen u.v.m. im Bereich der Mitarbeiter zu finden. Diese bereits bestehenden Aufgaben aller Mitarbeiter der Monopolkommission waren durch die Migration der Arbeitsplatzrechner und der Server auf Open-Source-Software unbehindert weiterzuführen.

1.2. Die Auswahl der Distribution

Im Rahmen der Bieterkonferenz zum Auftrag standen acht verschiedene Firmen vor der Aufgabe, den Projektverantwortlichen des BSI als auch der Monopolkommission eine Aussage über Präferenzen für die einzusetzende GNU/Linux-Distribution zu geben. Die überwiegende Mehrheit (7 von 8) der Unternehmensvertreter sprach sich für Debian aus, da das Thema Softwareverteilung und Stabilität für die PC-Arbeitsplätze (Clients) mit besonderer Bedeutung versehen war. Die automatische Verteilung von Software-Paketen ist in einem Netzwerk von Computern eine der Zeit sparenden Methoden, um die jeweiligen Arbeitsplatzrechner mit der aktuellsten Software zu versorgen. Bisher waren dafür teure Werkzeuge notwendig, oder die Aktualisierung und das Neuinstallieren von Software wurde mittels Datenträger (Disketten oder CDs) von einem qualifizierten Mitarbeiter am einzelnen Arbeitsplatz vorgenommen.

1.3. Ziel der Migration

Ziel der Server-Migration war ein harmonisch aufeinander abgestimmtes Funktionieren der Clients und Server, also eine Client/Server-Landschaft, welche flexibel einsetz- und erweiterbar sein sollte und die Gesichtspunkte Wartbarkeit, Verfügbarkeit und Wiederherstellbarkeit berücksichtigte. So wurden sämtliche Server-Komponenten bzw. -Dienste auf die Anforderungen der Clients angepasst, insbesondere hinsichtlich zentraler Administration, Datenhaltung, Update-Fähigkeit der Clients und dynamischer Benutzerkonfiguration. Bei der dynamischen Benutzerkonfigura-

tion werden dem Arbeitsplatzrechner erst bei der Anmeldung eines Benutzers (Users) bestimmte Konfigurationsparameter (z.B. ob ein Benutzer ein Zugriffsrecht auf ein CD-ROM-Laufwerk hat oder nicht) übergeben.

Im Feinkonzept wurde festgelegt, dass die Konfiguration der einzelnen Arbeitsplatzrechner ausnahmslos von zentraler Stelle über einen Administrationsserver erfolgt, also keine Administration (Einstellen von Druckertreiber, Grafikauflösung, auf dem Rechner vorhandene Software) lokal am Arbeitsplatz zu tätigen ist. Deshalb sollte auch das Administrationswerkzeug entsprechend einfach bedienbar sein, denn die Administratoren sollten ja nicht in kürzester Zeit zu Linux-Profis ausgebildet werden, sondern mit einfachen Mitteln alle notwendigen und wünschenswerten Aufgaben erledigen können. Ziel der Client-Migration war der Austausch der vorhandenen Applikationen durch gleichwertige OSS-Applikationen oder solchen, die auf dem OSS-Betriebssystem GNU/Linux lauffähig sind. Die Nutzer sollten sich dabei so einfach und so schnell wie möglich an die veränderte Umgebung gewöhnen und alle wesentlichen Funktionen ihres bisherigen Arbeitsplatzrechners wiederfinden. Dabei sollte eine „Vor-Ort-Administration" per qualifiziertem Mitarbeiter am jeweiligen Arbeitsplatz weitgehend unterbleiben können. Gleichzeitig sollten für die sensiblen Daten eine zentrale Datensicherung und eine den hohen Sicherheitsanforderungen der Monopolkommission gerecht werdende Zutrittsbeschränkung zu den Daten implementiert werden. Die Anschaffung von neuer Client-Hardware sollte soweit als möglich unterbleiben.

2. Das Konzept

2.1. Der Client als Basis

In den ersten zwei Wochen des Projektes trafen sich die Projektbeteiligten, um aus dem „2-Seiter" der Ausschreibungsunterlagen und dem angebotenen Konzept der ausführenden Firmen ein Feinkonzept zu erstellen, das die weiteren Arbeiten bestimmen sollte. Bereits in der Ausschreibung war definiert, dass die Migration eine Migration von so genannten FatClients[1] sein sollte, denn als Clients sollten die handelsüblichen PCs zum Einsatz kommen, die bereits seit ca. 2 Jahren im Einsatz waren. Die Clients selbst sollten die Applikationen vorhalten und ihre Prozessorleistung zur Ausführung dieser einsetzen. Auch die Druckaufbereitung, das Scannen von Daten, die Webcam u. v. m. sollten vom PC übernommen werden. Alle Applikationen der Monopolkommission, die auf dem Betriebssystem Windows 2000 lauffähig waren, wurden durch Open-Source-Software (OSS) oder OSS-lauffähige Applikationen ersetzt. Einzige Ausnahme ist eine von der Firma Hoppenstedt erstellte CD-ROM für Firmendaten, deren Applikation Einträge in eine Windows-Registry[2], der zentralen Datei für die Registrierung von Programmen im Windows-Betriebssystem, vornimmt, damit die Programme unter anderem alle Betriebssystemfunktionen nutzen können. Jedoch auch hier konnten die Erfahrungen der Mitarbeiter

1 FatClient ist ein voll ausgestatteter PC, im Gegensatz dazu ist ein ThinClient ein Gerät ohne Festplatte und lokale Applikationen.

2 Mehr zur Registry unter http://www.zdnet.de/downloads/weekly/7/weekly_143-wc.html.

eines anderen Pilotprojektes (Bundeskartellamt[3]) genutzt werden, um diese Applikation unter WINE[4] (einer Emulation, die auf einem Linux-System einer für Windows entwickelten Software eine entsprechende Umgebung zur Verfügung stellt) verfügbar zu machen.

Im Unterschied zu Windows 2000 stehen unter GNU/Linux verschiedene Fenstermanager zur Auswahl, die zum Einsatz kommen könnten. Bereits im Angebot wurde der IceWM[5] als Fenstermanager vorgeschlagen, da sich die Optik sehr an Windows 9x, NT und 2000 anlehnt und auch die gewohnten Tastaturbefehle, wie ALT-F4 für das Schließen eines Fensters, vorbestimmt sind. Als Applikation geht er auf dem Client moderat mit den Systemresscourcen, wie z.B. Prozessor oder Arbeitsspeicher, um. Die „Desktop"-Funktionalität, d. h., dass Programm-Icons auf der Oberfläche liegen und per Mausklick die damit verbundenen Programme gestartet werden können, wird nicht über den IceWM bereit gestellt, sondern durch den Filebrowser GNOME Midnight Commander[6].

Als Office-Applikation wurde Staroffice 6.0 ausgewählt, welches zwar auf dem Open-Source-Produkt Openoffice basiert, jedoch selbst nicht Open-Source-Software ist, da lizenzpflichtige Produkte wie eine Datenbank oder eine Rechtschreibprüfung hinzugefügt wurden. Ausschlaggebend für den Einsatz von Staroffice 6.0, alternativ zu Openoffice, war die Rechtschreibprüfung. Da in der Monopolkommission kein anderes Office-Produkt mehr vorhanden sein würde, kam einer möglichst guten Rechtschreibprüfung eine besondere Bedeutung zu. Einigen Benutzern wurde alternativ die Möglichkeit gegeben, wie bisher mit Latex[7], einem auf dem Textsatzsystem TeX[8] basierenden Werkzeug, um Texte zu verfassen und zu layouten, zu arbeiten.

Für das Suchen von Dateien steht mit dem Windows-Explorer ein Produkt im Windows-Umfeld zur Verfügung, das in der intuitiven Bedienbarkeit auch für den Einsatz in der Monopolkommission als Maßstab für Filebrowser-Funktionen benannt wurde. Der GNOME Midnight Commander (GMC) bietet ebenfalls Linuxbasiert die gewünschten Funktionen und ermöglicht darüber hinaus das „Öffnen" und „Ansehen" von Archiven (z.B. zip oder tar) sowie das Öffnen der darin befindlichen Dateien, ohne auf ein entsprechendes Archivierungswerkzeug zurückgreifen zu müssen.

Für die Mail-Funktionalität standen zu Beginn des Projektes drei OSS-Produkte zur Auswahl. KMail, Sylpheed und Mozilla. Da im Zusammenhang mit der Umstellung der Arbeitsplatzrechner auch die bereits vorgesehene digitale Signatur[9] zur eindeutigen und rechtssicheren Unterschrift von Dokumenten oder auch E-Mails eingeführt werden sollte, wurde ein einfach zu bedienender Mail-Client gesucht, der neben der leicht zu bedienenden grafischen Oberfläche auch die Unterstützung für

3 Vgl. http://www.bundeskartellamt.de.
4 Vgl. http://www.winehq.com/.
5 Vgl. http://www.icewm.org.
6 Vgl. http://www.ibiblio.org/mc/.
7 Vgl. http://www.dante.de.
8 Vgl. http://www-cs-faculty.stanford.edu/~knuth/.
9 Vgl. http://www.bund.de/Gut-zu-Wissen/Kommunikation-und-Medien/Internet/Elektronische-Signatur-.7482.htm.

GNUpg[10] als digitaler Signatur bot. Die Entscheidung für einen Standardarbeitsplatz fiel auf Sylpheed[11].

Als Browser standen ebenfalls einige Applikationen zur Auswahl: Konquerer, Mozilla, Galeon. Da die Funktionen zum Durchsuchen von Internetseiten mindestens den Umfang der bisher (MS Internet Explorer) zur Verfügung stehenden Applikation haben sollte, war Mozilla auf Grund der umfangreichen Funktionen auch in Bezug auf Plug-Ins, wie z.B. Macromedia Flash, schon ein Favorit als Browser, da jedoch die Mail-Funktionen von einer anderen Applikation übernommen würde, wurde bei den Testnutzern Galeon eingesetzt. Schnell fanden sich die Testnutzer zurecht, und somit wurde Galeon[12], der auf der „Rendering-engine", der Grafikdarstellungs-Software von Mozilla, basiert, gewählt.

Bisher kamen ein vom Scanner-Hersteller mitgeliefertes Programm Adobe Photoshop LE (Limited Edition) und der entsprechende Windows-Treiber zum Einsatz. Da für den Scanner ein Treiber für Linux bereitstand, wurde nur noch nach einer Scanner-Software unter Linux gesucht. SANE (Scanner Access Now Easy, Frontend Xsane)[13] wurde den Testbenutzern zur Verfügung gestellt, und nachdem die bisher genutzten Funktionen über SANE ebenfalls verfügbar waren, konnte sichergestellt werden, dass die Benutzer die erforderlichen Aufgaben erledigen können. Um eine leichte Weiterverarbeitung der Daten zu ermöglichen, wurde das Scannerprogramm auch zum Aufruf aus Staroffice und GIMP[14] eingerichtet. Dort kann man wie gewohnt unter dem entsprechenden Menüpunkt ein Bild scannen.

Für die Erstellung von pdf-Dokumenten wurde auf Basis des Betriebssystems Windows 2000 der Adobe Acrobat in der Version 5.0 eingesetzt. Von den verschiedenen Funktionen wurde im Wesentlichen der Acrobat Distiller benutzt, um aus doc-Dokumenten entsprechende pdf-Dokumente zu erstellen. Da die weitergehenden Funktionen des Adobe Acrobat nicht benutzt wurden, war es möglich, die im Staroffice 6.0 und Openoffice integrierte Funktion des Erstellens eines pdf-Dokumentes über den Druckdialog und die Auswahl des „PDF-Konverter" zu nutzen. Für das Öffnen von pdf-Dokumenten zum Lesen und Drucken steht der Acrobat Reader in der Version 5.0 als Freeware auch für Linux zur Verfügung.

Mit den hier beschriebenen Elementen war der Arbeitsplatz-PC in seiner Grundausstattung als Linux-PC erstellt. Dass die Arbeiten damit nicht abgeschlossen waren, sondern lediglich die Basis der Lösung erstellt wurde, zeigen die im Folgenden geschilderten Details der Lösung.

2.2. Integration besonderer Sicherheitsmerkmale am Arbeitsplatz

Als Wunsch der Monopolkommission stand von Anfang an fest, dass für die Anmeldung am Client eine Kombination von zwei Merkmalen zur Authentifizierung am Arbeitsplatz zum Einsatz kommen sollte, da die Kombination von zwei Si-

10 Vgl. http://www.gnupg.de/.
11 Vgl. http://www.sylpheed.org.
12 Vgl. http://www.gnome.org/gnome-office/galeon.shtml.
13 Vgl. http://www.sane-project.org/.
14 Vgl. http://www.gimp.de/.

cherheitsmerkmalen zu einer Erhöhung der Fälschungssicherheit einer Authentifizierung führt. Ergebnis war die Kombination von Chipkarte und Fingerabdruck.

Für die Erstellung von Chipkarten, zum Erfassen von Fingerabdrücken und zur Erstellung der Digitalen-Signatur-Daten wurde eine integrierte Applikation erstellt. Mit der Applikation kann man über eine grafische Oberfläche sehr leicht eine Chipkarte erstellen, die für die digitale Signatur und zur Authentifizierung bei der Anmeldung am Arbeitsplatz genutzt wird. Die Gewinnung der Biometriedaten wird dabei über ein Produkt der Siemens AG[15] durchgeführt, welches nicht quelloffen zur Verfügung steht. Die Daten, die per Minuzien-Verfahren[16] aus einem gescannten Fingerabdruck gewonnen werden, liegen verschlüsselt auf dem Webserver.

Die Chipkarte wird zur Authentifizierung in beiden möglichen Umgebungen („Online" und „Offline") auf gleiche Art, jedoch mit unterschiedlicher Ausprägung genutzt.

Unter der „Online"-Umgebung wird die Netzwerkumgebung verstanden, bei der folgende Voraussetzungen gelten. Zum einen muss ein Arbeitsplatzrechner eine physische Verbindung zum Netzwerk haben (Netzwerkkabel). Zum anderen müssen dann im Netzwerk folgende Dienste zur Verfügung stehen: Nameservice, Adminservice (LDAP-Service), Fileservice.

Für den Fall, dass die Netzwerkverbindung gestört ist und die Arbeitsplatzrechner keinen Zugriff zu den Servern haben, ist keine Netzwerkanmeldung am Arbeitsplatz möglich. Der Benutzer erkennt dies an der Anmeldemaske. Er findet dann nicht die gewohnte Anmeldemaske mit grünem Schriftzug vor, sondern eine spezielle „Offline"-Anmeldemaske mit rotem Schriftzug „offline".

In solchen Fällen kann sich ein Benutzer nur „offline" anmelden. Dafür wird ausschließlich die Chipkarte benötigt, da die Biometriedaten bei einer Anmeldung mit den Daten des LDAP-Directory verglichen werden müssten, die aber nur über das Netzwerk zu erreichen wären.

Als Client ist also auch ein Laptop einsetzbar, der ja nicht immer im Netz sein muss, der jedoch über die Administration zentral mit allen Diensten und Applikationen versorgt wird, sobald er sich im Netzwerk befindet.

2.3. Digitale Signatur

Im Rahmen der Migration der Monopolkommission sollte die Einführung der digitalen Signatur für den gesicherten Mailverkehr erfolgen. Da das BSI mit dem Projekt Ägypten[17] über ein Open-Source-Projekt verfügt, das den Standard nach Sphinx[18] zur Erstellung einer Public-Key-Infrastruktur (PKI) erfüllt, sollte wenn möglich die digitale Signatur auf Grund der dort getätigten Entwicklung mit dem Mail-Client KMail umgesetzt werden. Da zum Entscheidungszeitpunkt die Arbeiten im Projekt Ägypten noch nicht in Form eines einsatzreifen Produktes abgeschlossen

[15] Seit 1.10. 2003 hat die Firma Bromba GmbH (http://www.bromba.de) die Technik von der Siemens AG übernommen.

[16] Vgl. http://www.fraunhofer.de/german/publications/df/df1998/198-30.htm.

[17] Vgl. http://www.bsi.de/aufgaben/projekte/sphinx/aegypten/opensour.htm.

[18] Vgl. http://www.bsi.de/aufgaben/projekte/sphinx/index.htm.

waren, wurde die digitale Signatur in Form von GNUpg-Schlüsseln realisiert. Dabei werden die GNUpg-Daten verschlüsselt auf der Chipkarte abgelegt.

2.4. Die Serverumgebung

Das Konzept beinhaltete von Beginn an einen zentralen Ansatz für die wesentlichen Komponenten. Erst durch die zentralen Komponenten, die auf den reibungslosen Einsatz der Linux-PC-Arbeitsplätze abgestimmt wurden, konnte die leicht zu administrierende Umgebung die gesteckten Ziele (einfache Administration bei höherer Sicherheit der Arbeitsumgebungen) erreichen. Dabei wurden die grundsätzlich auf verschiedenen Servern liegenden Funktionen so angepasst, dass sie sich im Rahmen des Projektes auch auf einem einzigen Server installieren lassen.

Bei der INFRAppliance Administration Console[19] handelt es sich um ein webbasiertes Werkzeug, mit dem alle wesentlichen Parameter für die Server, Clients, Applikationen und Benutzer administriert werden können. Die Administration Console unterstützt in der IT-Umgebung die zentrale Verwaltung von

- Benutzerkonten und -einstellungen
- Druckern
- Softwarepaketen für die Softwareverteilung und der
- Client-Hardwarekonfiguration.

Damit wurde die Administration Console Angelpunkt für die umfangreichen zentralen Funktionen, die in dem Konzept zur Verfügung stehen. Der INFRAppliance Fileserver[20] speichert in der Monopolkommission alle von den Benutzern erstellten Daten in deren „Home"-Verzeichnissen ab. Gleichzeitig sind für den Austausch von Daten oder das Zusammenarbeiten von Mitarbeitern Gruppenverzeichnisse angelegt, in die die Mitglieder einer Gruppe Dateien legen und aus denen sie sie auch lesen können. Integriert ist ein Backup-Modul, mit dem die Daten des Fileservers gesichert und rückgesichert werden können.

Der INFRAppliance Webserver[21] hat verschiedene Aufgaben. Zuerst dient er als Webserver für die zentralen Webapplikationen, die im Rahmen des Projektes entstanden sind. Zusätzlich werden die Intranetseiten der Monopolkommission über den Webserver zur bereitgestellt. Im Zusammenhang mit der Softwareverteilung stellt der Webserver die Softwarepakete zur Verfügung. Für die Webapplikationen jedoch auch für andere Applikationen sind auch die Datenbanken (PostgreSQL, mySQL) hier implementiert. Die drei Server und die ergänzenden Module wurden innerhalb des Projektes so erstellt bzw. ergänzt, dass die gesamte Rechnerlandschaft (Clients und Server) von der Administration Console aus zu administrieren ist. Dabei basiert die Oberfläche der Administration Console auf der Open-Source-Software (OSS) Webmin[22]. Die Weboberfläche gestattet die Administration der Clients und Server unabhängig vom Standort des Administrators von jedem im Netzwerk befindlichen Arbeitsplatzrechner.

---

19 Vgl. http://www.sfi-director.org.
20 Vgl. http://www.sfi.ch.
21 S. o.
22 Vgl. http://www.webmin.de/.

Als zentraler Verzeichnisdienst kommt openLDAP zum Einsatz, wo letztlich alle Einstellungen, die zu Benutzern, Software oder Hardware administriert werden, abgelegt werden. Damit sind die Daten für die Wiederherstellung von Benutzerprofilen besonders leicht zu erhalten. Gleichzeitig werden die Benutzerprofile auch in den jeweiligen „Home"-Verzeichnissen der Benutzer hinterlegt, sodass bei der Anmeldung eines Benutzers die entsprechenden individuellen Einstellungen für den beliebigen Arbeitsplatz zur Verfügung stehen.

2.5. Zentraler Ansatz

Für die Kosten des Betriebs einer IT-Infrastruktur sind eine ganze Reihe von Faktoren von Bedeutung. Neben den Lizenzkosten für Betriebssysteme und Applikationen auf Servern und Clients sind die laufenden Kosten des Betriebs der wesentliche Faktor.

Der zentrale Ansatz wurde grundsätzlich für alle Funktionen zum Betrieb der IT-Struktur in der Monopolkommission und für die zu erstellenden Webapplikationen gewählt, da erst dadurch die Administrationskosten wesentlich gesenkt werden können. Wenn Administration vor Ort entfallen kann, die Benutzer grundsätzlich nicht in der Lage sind, die auf dem LDAP-Server liegenden Einstellungen zu ändern, und die aktuellen Konfigurationen jederzeit eingesehen werden können, d. h., das System jederzeit definiert ist, und letztlich auch noch die „Fernwartung" über systemeigene Mechanismen mit Linux möglich ist, dann kommt mit der zentralen Administration und Datenhaltung bei dezentraler Verfügbarkeit von Rechenleistung lokal am Arbeitsplatz derzeit ein Optimum an Kosten senkenden Faktoren zusammen.

Die Benutzerdaten (z.B. Name, User ID, Group IDs) und Benutzereinstellungen für Applikationen werden über die Administration Console angelegt und administriert. Ebenfalls werden weitere Einstellungen der Hardwarekonfiguration der Clients über die Administration Console auf dem LDAP-Server[23] abgelegt. Die Daten werden automatisch bei Installation der Clients oder über das entsprechende Modul der Administration Console manuell registriert. Die vom Client automatisch erkannten Hardwaredaten (Grafikkarte, Grafikkartentreiber und Monitoreinstellungen) werden gespeichert.

Der Administrationsserver stellt auch die Verwaltung der für die Clients vorgesehenen Softwarepakete zur Verfügung. Die Daten der Benutzer werden zentral auf dem Server im INFRAppliance Fileserver mit Backup gehalten. Damit ist gewährleistet, dass die sensiblen Daten in das bestehende Sicherungskonzept für den Server eingebunden werden und eine Delegierung der Verantwortlichkeiten der individuellen Sicherung der Daten auf dem Client – mit den damit verbundenen Schwierigkeiten der Kontrolle – unterbleiben kann.

Die zentrale Softwareverteilung ermöglicht die automatische, schnelle und einfache Verteilung von Betriebssystem-Updates, Applikationen und Security-Patches. Dabei wird die zu verteilende Software in systemkonforme Pakete für die Debian-Distribution gepackt, die den Clients zur automatischen Installation zur Verfügung gestellt werden. Beim Starten des Rechners werden der Softwarestand des Clients

[23] Vgl. http://verzeichnisdienst.de/ldap/.

überprüft, und eventuell zur Verfügung stehende Softwarepakete automatisch heruntergeladen und installiert. Die Möglichkeit, auch manuell Softwarepakete auf die Clients zu laden, bleibt dabei erhalten.

2.6. Spezielle Aufgaben

Neben der für den Client und auf der Serverseite entwickelten Lösung sollten im Rahmen der Migration noch weitere Funktionalitäten erstellt oder aus dem Microsoft-Umfeld ersetzt werden.

Da eine Reihe von Funktionen von mehreren Arbeitsplätzen aus genutzt werden sollte, entstanden im Rahmen das Projektes Webapplikationen, die allen Mitarbeitern der Monopolkommission per Webbrowser zur Verfügung stehen.

2.6.1. Bibliotheksverwaltung

Da die Monopolkommission eine eigene Bibliothek besitzt, wurde zur Verwaltung des Bücherbestandes der Bibliothek ein Katalog mit Weboberfläche erstellt, mit dem jeder berechtigte Mitarbeiter von seinem Arbeitsplatz aus die Liste der verfügbaren Bücher einsehen und nach Büchern mit bestimmten Attributen durchsuchen kann. Die Weboberfläche bietet dabei für Mitarbeiter mit Verwalterbefugnissen zusätzlich die Möglichkeit, neue Bücher aufzunehmen, die Daten bereits aufgenommener Bücher zu verändern oder aus dem Katalog zu entfernen.

Folgende Funktionalitäten stehen dem Benutzer demnach über die Intranet- Seite zur Verfügung:

- Suche nach Büchern
- Anzeige der Neuerwerbungen
- Hinzufügen eines neuen Buches (falls der Benutzer der Gruppe angehört, der die Verwalterbefugnisse eingeräumt wurden)
- Anzeige der Ausleihen eines Benutzers.

Dem Administrator der Bibliotheksverwaltung stehen weitere Funktionen zur Verfügung, die ihm die Verwaltung der Buchbestände und der Ausleihen durch Mitarbeiter der Monopolkommission ermöglichen.

2.6.2. CD-Archivierung

Bei der Monopolkommission wurden abgeschlossene Projekte, z.B. in der Vergangenheit erstellte Gutachten, auf CD archiviert.

Zur Verwaltung dieser CDs wurde ein CD-Katalog mit Weboberfläche erstellt, mit dem jeder berechtigte Mitarbeiter von seinem Arbeitsplatz aus die Liste der verfügbaren CDs einsehen und nach Dateien mit bestimmten Attributen (Name, Änderungsdatum, Größe) durchsuchen kann. Die Weboberfläche bietet für Mitarbeiter mit Editorbefugnissen ferner die Möglichkeit, beliebige neue CDs in den Katalog aufzunehmen sowie die Daten von existierenden CDs zu modifizieren bzw. aus dem Katalog zu entfernen. Folgende Funktionen wurden in der Applikation verwirklicht:

- Durchsuchen des CD-Bestands nach diversen Kriterien (z.B. nach Kategorie oder Entstehungsjahr)
- Suche im CD-Bestand nach CD-Attributen
- Suche nach Dateien (Ermitteln der CDs, die bestimmte Dateien enthalten)

- Hinzufügen einer neuen CD (falls der Benutzer der Gruppe angehört, der die Editorbefugnisse eingeräumt wurden).

2.6.3. Kalender

Vor der Migration wurde in der Monopolkommission kein elektronischer Kalender mit seinen verschiedenen Möglichkeiten, z.B. der Terminabstimmung, eingesetzt. Im Rahmen der Migration wurde zur Abstimmung von Terminen mit der PHP-Groupware[24] ein Terminkalender per Weboberfläche zur Verfügung gestellt, der über die Intranetseite durch die Benutzer aufgerufen werden kann. Hier können die Mitarbeiter ihre eigenen Termine und Gruppentermine eingeben und organisieren. Dabei verfügt der Terminkalender über die Funktion, Termine mit anderen Mitarbeitern abgleichen zu können, d. h. Überschneidungen entsprechend anzuzeigen.

2.6.4. Adressbuch

Die Aufgabenstellung im Zusammenhang mit dem Adressbuch war geprägt von unterschiedlichen Anforderungen und unterschiedlichsten Datenquellen für die Mitarbeiter der Monopolkommission. Der Informationsverbund der Bundesbehörden (IVBB[25]) sollte berücksichtigt werden – genauso wie der hauseigene Adressbuchansatz des Bundeskartellamtes. Auch hier wurde im Rahmen des Feinkonzeptes entschieden, den Anforderungen durch die Entwicklung einer Webapplikation zu entsprechen. Zum späteren Zeitpunkt wurde die Integration der Funktionen in PHP-Groupware beschlossen und realisiert. Den Mitarbeitern steht damit eine Auswahl verschiedener „Adressbücher" zur Verfügung, die über die Datenbankanbindung von Staroffice auch für Serienbrieffunktionen genutzt werden kann.

2.6.5. Migration Fragebogen-Workflow und Statistikdatenauswertung

Die Vorgaben zu diesen beiden Punkten waren im Rahmen der Ausschreibung und auch bei der Erstellung des Feinkonzeptes im Wesentlichen auf die Anforderungen der beiden betroffenen Mitarbeiter zugeschnitten. Für beide Teilprojekte gab es keine von vornherein fest definierten Applikationen, die migriert werden mussten. Stattdessen stützten sich die Aufgaben, die von den wissenschaftlichen Mitarbeitern im Zusammenhang mit der Auswertung von Statistikdaten für die Gutachten und im Zusammenhang mit der Datenerhebung mittels an Firmen gesandter Fragebögen zu erledigen sind, im Wesentlichen auf Funktionen, die von dem Softwareprodukt Microsoft Office abgedeckt wurden. Dabei wurden vor allem dessen Teilkomponenten Access (Datenbank), Excel (Tabellenkalkulation) und Visual Basic (Makro- bzw. Skriptsprache zur Prozessautomatisierung) eingesetzt. Da das im Zuge der Migration als Ersatz für Microsoft Office eingeführte Staroffice diese Funktionalitäten nicht vollständig anbietet – so ist zwar mit StarCalc eine mit Excel vergleichbare Tabellenkalkulation vorhanden, es fehlen jedoch die Datenbank und eine zu Visual Basic vollständig kompatible Programmiersprache –, musste für die fehlenden Funktionalitäten anderweitig Ersatz beschafft werden. Es war also notwendig, die Aufgaben der beiden Mitarbeiter zu analysieren, wiederkehrende Problem-

[24] Vgl. http://phpgroupware.org/.
[25] Vgl. http://www.ivbb.de.

stellungen zu identifizieren und dann Methoden und Werkzeuge auszuwählen, mit denen die Mitarbeiter ihre Aufgaben, unter anderem der Datenauswertung und -zusammenstellung, auch nach der Umstellung auf Linux und Open-Source-Software würden erledigen können. Zuletzt waren die Mitarbeiter in der Anwendung dieser Methoden und Werkzeuge zu unterweisen.

Da sich die Funktionalitäten von Staroffice, wie oben bereits angedeutet, auf die grundlegenden Office-Bereiche Textverarbeitung, Tabellenkalkulation und Präsentationserstellung konzentrieren, war zunächst Ersatz für die Datenbankkomponente von MS Access zu finden (die Datenzugriffskomponente von Access, also Funktionen zum Zugriff auf und zur Arbeit mit externen Datenquellen, ist in Staroffice überwiegend vorhanden). Hierbei fiel die Wahl wiederum auf PostgreSQL – aus denselben Gründen, aus denen dieses relationale Datenbanksystem auch für die Webapplikationen ausgewählt worden war und um nicht unnötigerweise ein zusätzliches System einzuführen. Als Programmiersprache wurde wie bei den Webapplikationen Perl gewählt, wobei hier die herausragenden Fähigkeiten zur Textextraktion (wichtig bei der Verarbeitung von Daten aus externen Quellen), die Vielfalt an hochwertigen und frei verfügbaren Modulen für alle denkbaren Anwendungszwecke, die gute Datenbankanbindung durch das Datenbankinterface DBI sowie die Möglichkeit zur objektorientierten Programmierung und strukturierten Fehlerbehandlung (Exceptions) den Ausschlag gaben. Dazu kamen noch diverse kleinere Open-Source-Applikationen und Hilfsprogramme, um verschiedene spezifische Anforderungen der beiden Teilbereiche abzudecken, z.B. Software zur Datenvisualisierung (Ploticus) sowie diverse Datenbank-Hilfsprogramme. Mit dem Open-Source-Tool pgadminII und dessen „Database Migration Wizard“ stand ein komfortables Hilfsmittel zur Verfügung, um bestehende Datenbanken von MS Access nach PostgreSQL zu migrieren.

Die beiden Teilprojekte wurden also in einem interaktiven Prozess, d. h. in direkter Zusammenarbeit mit den betroffenen Mitarbeitern, angegangen. Hierbei war von Vorteil, dass beide Mitarbeiter bei ihrer Arbeit mit dem bisherigen System bereits an prinzipbedingte Grenzen gestoßen waren, deren Überwindung sie sich als Nebeneffekt der Migration erhofften, sodass ihre Motivation und Bereitschaft zur Kooperation entsprechend hoch waren und das angestrebte Ziel weitgehend erreicht werden konnte.

3. Der Ablauf der Migration

Bereits bei der Bieterkonferenz waren Mitarbeiter, also spätere „User“ der Lösung, anwesend. Im Rahmen des Projektes wurden sehr schnell (innerhalb der ersten vier Wochen) einzelne Mitarbeiter mit Linux-Arbeitsplätzen ausgestattet, die von den Applikationen her dem finalen Client sehr nahe kamen. Dabei konnten die Mitarbeiter die Applikationen sehr früh testen und entsprechende Anregungen geben. Die im Rahmen des Feinkonzeptes ausgewählten Applikationen waren bereits sehr auf die Eignung hin geprüft und ausgewählt worden. Die gewählten Sicherheitsmechanismen des Einsatzes der Biometrie und der Chipkarte waren der deutlichste Beleg für die Umstellung an sich.

Die eigentliche Umstellung der Arbeitsplätze (Clients) und der Serverumgebung konnte auf Grund der vielen Vorarbeiten erst sehr spät im Projektablauf erfolgen, genau gesagt, zwei Wochen vor Projektende. Die Migration der Serverumgebung ging dabei in zwei Schritten vonstatten. Zuerst wurden ein INFRAppliance Fileserver und ein INFRAppliance Webserver in der Testumgebung aufgestellt. Der bisherige Fileserver wurde parallel im Netz betrieben.

Die Mitarbeiter wurden aufgefordert, ihre Daten, die eventuell noch lokal auf den PC-Arbeitsplätzen vorhanden waren, auf dem zentralen Fileserver abzulegen. Für den eigentlichen Migrationstag wurde ein Freitag ausgesucht, da die Mitarbeiter im Zusammenhang mit der Migration zum Erfassen der Fingerprintdaten einbestellt werden mussten. Die mit Fingerabdruck erfassten Mitarbeiter konnten daraufhin an ihren Arbeitsplätzen arbeiten. Für die Mitarbeiter wurde eine kurze Einweisung für die Anmeldung mit Chipkarte und Biometrie gegeben. Am folgenden Montag standen Mitarbeiter von natural computing zur Verfügung, um eventuell auftretenden Problemen schnell begegnen zu können. Gleichzeitig setzten die Benutzerschulungen für die Mitarbeiter in ihren speziellen Aufgabenfeldern ein. Ursprünglich wurde die Schulung der Mitarbeiter in Gruppen vorgesehen. Auf Grund unterschiedlichster Anforderungen der einzelnen Mitarbeiter wurde jedoch auf die individuelle Schulung der Mitarbeiter für eine ganze Reihe von Funktionen umgestellt. Dies kam bei den Mitarbeitern gut an, ließ sich jedoch nur umsetzen, weil einige Mitarbeiter der Monopolkommission nicht vor Ort waren. Dabei wurde ein modularer Aufbau für die Schulung gewählt. Es wurden kleine Abschnitte gewählt, um dem Mitarbeiter Zeit fürs Tagesgeschäft zu lassen, und die Mitarbeiter konnten mitentscheiden, welche Module von ihnen benötigt wurden. Die wissenschaftlichen Mitarbeiter brauchten zum Beispiel keine Formatierungsdetails für Texte, da eine Kollegin die Formatierung der „offiziellen“ Texte übernimmt, zum anderen wurde nur für das Sekretariat der Umgang mit Serienbriefen als wichtig eingestuft. Der Aufwand der Schulungen wuchs jedoch damit auch an. Zusätzlich wurde die Übernahme der „Altdaten“ aufwändiger als erwartet, da die Datenvoluminen der einzelnen Arbeitsplatzrechner („Laufwerk C“)im Vorfeld nur geschätzt werden konnten.

Erst nachdem die Clients migriert waren, wurde der bisherige Fileserver zum zentralen INFRAppliance Server mit den oben genannten Funktionen. Da für die Migration des Fileservers keine Testmöglichkeit existierte, entschloss man sich, eine Woche nach der Client-Migration an einem Sonnabend auch den Server zu migrieren. Für den Fall, dass dabei Probleme auftreten würden, hätte man auf die bereits vorhandene INFRAppliance-Umgebung in der Testumgebung zurückgreifen können. Doch die Migration lief im Wesentlichen problemlos, sodass bis auf die Backup-Funktion alle Funktionalitäten zur Verfügung gestellt wurden.

Das Backup-Problem wurde noch mal stark in den Vordergrund gerückt, als bei einem Backup-Versuch zum Wochenende der Server mit der verwendeten Redhat-Version abstürzte. Das Aufspielen eines neuen Kernels behob die Probleme des Betriebssystems mit der zur Verfügung stehenden Hardware. Seitdem läuft der Server im Produktivsystem.

Die Nutzer arbeiten bereits seit dem 28. Oktober 2002 im Produktivsystem. Die bei der Anmeldung für den E-Mail-Verkehr zur Verfügung gestellte GnuPG-Signa-

tur wird von der Chipkarte über die per USB angeschlossene Tastatur dem Client zur Verfügung gestellt. Dieser Vorgang dauert ca. zehn bis 15 Sekunden. Nicht jeder Mitarbeiter war mit dieser Komforteinbuße zufrieden.

Das Projekt beinhaltete auch den Support der Lösung bis zum 30. Juni 2003. In dieser Zeit standen den Mitarbeitern Hotline- und E-Mail-Support zur Verfügung sowie eine Anzahl von Tagen vor Ort, um eventuell auftretende Probleme und Anforderungen bearbeiten zu können. Die Mitarbeiter nahmen alle Arten des angebotenen Supports wahr. Gegenstand der Supportanfragen war dabei nur zum kleineren Teil auf das System an sich bezogen, zum größeren auf Funktionalitäten aus den eingesetzten Applikationen.

Selbstverständlich konnte auch während des Projektes nicht jeder Prozess in der Monopolkommission so untersucht und aufgenommen werden, wie man sich das wünschen würde, dazu stand für das gesamte Projekt viel zu wenig Zeit zur Verfügung. Also kamen im Rahmen des Supports auch Dinge ans Licht, die nicht berücksichtigt waren. Doch die Anfragen konnten in der Regel sehr schnell und auch fast durchweg mit einer Lösung beantwortet werden.

Die Anfrage nach speziellen Programmen, die einzelne Mitarbeiter einsetzen wollten, konnte nicht immer positiv beantwortet werden, da leider nicht jede Applikation für Linux existiert bzw. nicht jede Applikation mithilfe von Emulatoren unter Linux funktioniert. Doch konnten einige Alternativen angeboten werden.

Was interessanterweise nicht zu einer Supportanfrage führte, war zu Beginn des Jahres der Stromausfall im Haus der Monopolkommission. Die Systeme starteten nach dem Stromausfall ohne Probleme.

Die nächsten Schritte sind die Implementierung der VPN-Lösung (Virtual-Private-Network), mit der die Mitarbeiter über eine sichere Verbindung auch von außen über das Internet auf ihre Daten in den Räumen der Monopolkommission zugreifen können, das Update der Office-Umgebung oder eine Implementierung der Funktion „Faxen aus Staroffice“ oder anderen Applikationen heraus.

4. Fazit

Der Autor sieht sich hier nicht im Stande, die Zufriedenheit aus Sicht der Benutzer wiederzugeben. Diese können nur die Mitarbeiter der Monopolkommission selbst darstellen. Jedoch können die Häufigkeit der Anfragen und die Tiefe der Fragestellungen einen sicheren Hinweis darauf geben, dass die Migration der Monopolkommission auf Open-Source-Software (OSS) gelungen ist.

Die geforderten Funktionen konnten mit nachvollziehbarem Aufwand erstellt werden. Dabei lag der Schwerpunkt der Umstellung im Wesentlichen in der Abstimmung von vorhandenen Open-Source-Software-Komponenten, der Erstellung der zentralen Administrationsmöglichkeiten und der Implementierung der Sicherheitsfunktionalitäten sowie der Ablösung von Funktionen, die über die Applikation MS Access abgewickelt wurden.

Die größte Herausforderung bestand jedoch in der zeitlichen Definition der Arbeiten innerhalb von nur drei Monaten. Einige Funktionen wurden neu entwickelt

und angepasst, und solche Arbeiten waren mit einem extrem engen Zeitplan nur unter größten Schwierigkeiten zu bewerkstelligen.

Eine der Veränderungen, die den Mitarbeitern die neue Arbeitsumgebung täglich präsent werden lassen, ist die Integration der Chipkarte und Biometrie-Anmeldung. Die Funktionalität ist in der Art der Umsetzung leicht zu bedienen und tritt im Tagesgeschäft nicht störend zu Tage.

Das Ergebnis der Arbeiten ist eine in fast allen Bereichen unter der *General Public License* (GPL) stehende Arbeitsumgebung, die die vor der Migration vorhandenen Funktionalitäten wieder zur Verfügung stellt und an zahlreichen Punkten durch zusätzliche Funktionen ergänzt.

Die Ergebnisse der Migration wurden nach Abschluss der Arbeiten bei der Erstellung des Migrationsleitfadens durch die Koordinierungs- und Beratungsstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung (KBSt.) verwendet.

Bei dem Projekt ist herauszuheben, dass es sich zum ersten Mal um die Komplettmigration einer Bundesbehörde handelt, die exemplarisch nachweisen sollte, dass auch eine komplette Arbeitsumgebung auf Open-Source-Software migriert werden kann. Doch sollte man nicht übersehen, dass auf Grund der außerordentlichen zeitlichen Restriktionen nicht jeder Prozess in der Monopolkommission simuliert werden konnte und auch zukünftige Entwicklungen nur in begrenztem Maß einbezogen werden konnten. So ist nicht auszuschließen, dass in Zukunft die eine oder andere Applikation, die nicht für Linux zur Verfügung steht, auf einer anderen Plattform genutzt werden muss.

Die erstellte Lösung ist geeignet, bei gleich gelagerten Anforderungen für eine schnelle Implementierung einer Open-Source-Software-Lösung zu dienen. Bei andersartigen Anforderungen kann sie als Basis dienen, um Implementierungsschwellen von Open-Source-Software (OSS) zu senken oder Migrationsprojekte zu beschleunigen.

Eine weitere Implementierung für 95 Arbeitsplätze hat inzwischen beim Landesrechnungshof Mecklenburg-Vorpommern stattgefunden. In 15 Arbeitstagen konnte dort die Lösung mit einigen Anpassungen implementiert werden. Die Umstellung der Clients und die Schulung der Mitarbeiter auf die neuen Arbeitsplätze wurden von den Mitarbeitern des Landesrechnungshofes in Eigenregie durchgeführt und dauerten rund 6 Wochen.

Vorstellung Natural Computing GmbH:

Die Natural Computing GmbH wurde 2001 von fünf Personen mit unterschiedlichen Schwerpunkten im Bereich der IT-Branche gegründet, um mit dem Fokus auf den Einsatz von Linux am Arbeitsplatz am Markt tätig zu sein. Langjährige Erfahrungen aus verschiedensten Bereichen von IT-Systemen und Dienstleistungen wurden gebündelt, um Kunden integrierbare Lösungen auf Basis von Debian GNU/Linux anzubieten.

Natural Computing bietet seinen Kunden aus Mittelstand und öffentlicher Beratung die Entwicklung sowie Implementierung von Linux-Arbeitsplatzlösungen.und

ist Mitglied des LIVE Linuxverband e.V., der Autor ist Mitglied des Vorstandes des LIVE.

Open-Source-Software am Büroarbeitsplatz: Erfahrungen der Endanwender aus der Migration der Geschäftsstelle der Monopolkommission

KERSTIN TERHOEVEN

1. Einleitung

Nachdem Linux eine beachtliche Verbreitung im Serverbereich erreicht hat, wird nun zunehmend der professionelle Einsatz von Open-Source-Software auf dem Arbeitsplatzrechner diskutiert und praktisch umgesetzt. Ein Referenzprojekt, welches unter anderem der Erprobung der quelltextoffenen Software im Clientbereich (auf dem PC-Arbeitsplatz, sog. Clientbereich) diente, war die Migration der Geschäftsstelle der Monopolkommission. Im Folgenden werden daher die Motivation der Migration sowie die mit der Umstellung von proprietärer auf Open-Source-Software verbundenen längerfristigen Vor- und Nachteile aus Sicht der Nutzer dargestellt. Auf Basis dieser Betrachtungen werden schließlich Empfehlungen hinsichtlich der im Rahmen einer Entscheidung über die Umstellung von Arbeitsplatzrechnern auf Open-Source-Software zu berücksichtigenden Faktoren ausgesprochen.

2. Rahmenbedingungen der Migration

2.1. Motivation

Im Juni 2002 wurde seitens des Bundesinnenministers die Initiative ergriffen, Open-Source-Projekte in Bundesbehörden durchzuführen, um die Einsatzfähigkeit von Open-Source-Software und im speziellen Linux zu evaluieren. Das Bundesamt für Sicherheit in der Informationstechnik (BSI) wurde daraufhin von der Projektgruppe Software-Strategie im Bundesinnenministerium beauftragt, im Rahmen des Anti-Terror-Programms entsprechende Projekte zur Verbesserung der IT-Sicherheit durchzuführen.

Mit dem Modellprojekt des BSI ergab sich für die Geschäftsstelle der Monopolkommission die Möglichkeit, kostenlos eine Umstellung auf Open-Source-Software vornehmen zu lassen. Die Mitarbeiter der Geschäftsstelle der Monopolkommission, die bis dahin mit einem Windows-2000-Netzwerk gearbeitet hatten, erklärten sich bereit, im Rahmen eines Pilotprojektes eine entsprechende Umstellung durchführen zu lassen. Hiermit wurde die Erwartung verknüpft, die Anfälligkeit gegenüber Computerviren und -würmern zu verringern und Kosten für Softwarelizenzen zu sparen.

2.2. Projektverlauf

Gegenstand des Auftrages war die Migration der Arbeitsplätze, der verwendeten Applikationen und der vorhandenen Server auf Open-Source-Software. Als Linux-Distribution wurde Debian gewählt, da von der Projektleitung der Softwarevertei-

lung und Stabilität der PC-Arbeitsplätze eine besondere Bedeutung zugemessen wurde. Die Anmeldung der Benutzer an ihrem Arbeitsplatz-Computer erfolgt nach vollzogener Migration über Chipkarte und Fingerabdruckscanner.

Der Projektleitung gehörten neben einer Mitarbeiterin des BSI ein Mitarbeiter des Bundeskartellamtes sowie ein in leitender Funktion tätiger Vertreter der beauftragten EDV-Dienstleister an. Mitglieder und Mitarbeiter der Monopolkommission waren an der Projektleitung nicht beteiligt. Die Abnahme des letzten Teilprojektes der Migration erfolgte am 22. Oktober 2002. Auf Grund der detaillierten Vorbereitungen war die Umstellungsphase nicht mit einer wesentlichen Beeinträchtigung des Tagesgeschäfts verbunden.

Im November 2002 fanden Schulungen der Endanwender statt, die – abgestimmt auf den bestehenden Informationsbedarf – hauptsächlich eine Einführung in das Textverarbeitungsmodul von Staroffice beinhalteten. Die Schulungen waren Bestandteil des Migrationsvertrages. Der Umfang der Staroffice-Schulungen betrug pro Anwender ca. acht Stunden. Einzelne Endanwender wurden mit den grundlegenden Befehlen auf Betriebssystemebene sowie mit ausgewählten Funktionen des Systemmonitors Gtop[1] und des Fenstermanagers IceWM[2] vertraut gemacht. Darüber hinaus wurden in Einzelgesprächen individuelle Probleme erörtert und betriebssystemspezifische „Tricks und Kniffe“ weitergegeben.

Wegen des noch nicht ausreichend vorhandenen Expertenwissens im Haus, verbliebener Umstellungsprobleme und der geringen Erfahrung der Mitarbeiter mit den neu zu nutzenden Applikationen bestand nach der Migration ein dringender Bedarf an externer Beratung. Daher wurde mit dem Dienstleister im Anschluss an die Umstellung ein Beratungsvertrag abgeschlossen. Dieser beinhaltet für die Mitarbeiter der Monopolkommission die Möglichkeit, telefonisch oder per E-Mail Fragen zu stellen und Probleme darzulegen. Weiterhin ist ein Vor-Ort-Service der EDV-Experten in begrenztem Zeitumfang vereinbart worden.

2.3. EDV-Anwendungen und -Anwender

EDV-Anwendungen in der Geschäftsstelle der Monopolkommission bestehen hauptsächlich in der Erstellung von Schriftstücken, der Informationsrecherche im Internet, der Kommunikation per E-Mail sowie der Erfassung und Auswertung empirischer Daten. Vorrangig werden daher Programme zur Textverarbeitung, E-Mail-Programme und Browser sowie Datenbankapplikationen verwendet. Die Nutzung der Applikationen geschieht mittels einer grafischen Schnittstelle zum Betriebssystem. Weitere regelmäßige EDV-Anwendungen bestehen in der Erzeugung von Dateien im Adobe-Portable-Document-Format (pdf), der Tabellenkalkulation, dem Zugriff auf ausschließlich Windows-kompatible CD-ROM sowie der Dateisuche und -organisation.

Tabelle 1 gibt eine Übersicht der vor und nach der Migration in der Geschäftsstelle genutzten Applikationen sowie der Intensität, mit der die Applikationen unter GNU/Linux genutzt werden.

---

1 Mittlerweile bekannt als GNOME System Monitor – s. http://www.gnome.org/softwaremap/projects/procman/.

2 Vgl. http://www.icewm.org.

Funktion	*Applikationen unter Windows 2000*	*Applikationen unter GNU/Linux*	*Nutzungsintensität nach Migration*
Fenstermanager	MS Windows Desktop	IceWM	täglich
Textverarbeitung	MS Word	Staroffice	täglich
E-Mail	MS Outlook	Sylpheed, Mozilla	täglich
HTML-Browser	MS Internet Explorer	Galeon, Mozilla	täglich
Erstellen von pdf-Dateien	Adobe Acrobat	GNU Ghostscript via Staroffice	regelmäßig
Tabellenkalkulation	MS Excel	Staroffice	regelmäßig
Lesen von pdf-Dateien	Adobe Acrobat Reader	Adobe Acrobat Reader	regelmäßig
Dateisuche	Windows Suchassistent	GNOME Midnight Commander	regelmäßig
Zugriff auf Windows-kompatible CD-ROM	nicht erforderlich	Wine Desktop	regelmäßig
Bildbearbeitung	Imaging für Windows, Adobe Photoshop LE	GIMP, Image Magick	regelmäßig
Systemmonitor	Windows Taskmanager	Gtop	regelmäßig
Scannen	Adobe Photoshop LE	SANE	selten
Videokonferenzen	Netmeeting	GNOMEmeeting	nicht genutzt
CD brennen	Nero, WinOnCD	CD-Roast	selten
Textsatz		KTeXMaker2	selten
Packprogramm	WinZIP	Fileroller	selten
Datenerfassung und -auswertung	MS Access	PostgreSQL, Perl, PgAccess, Staroffice, phpPgAdmin	selten, Migration nur teilweise vollzogen
Webseitenerstellung	Netobject Fusion	AMAYA	nicht genutzt

Tabelle 1: Ersetzung und Nutzungsgrad der Applikationen

Zu der Geschäftsstelle der Monopolkommission gehören zwölf Mitarbeiter. Diese haben als Verwaltungsangestellte, Juristen und Ökonomen keine professionell vertieften EDV-Kenntnisse. Anwendungswissen wurde in der beruflichen Praxis oder im Rahmen von Schulungen erworben. Erfahrungen bestanden vor der Migration mit den Betriebssystemfamilien DOS, Windows, MacOS und vereinzelt mit Linux.

3. Beurteilung der eingesetzten Open-Source-Software

Eine ausführliche Darstellung der Vor- und Nachteile der quelltextoffenen Applikationen gegenüber den ursprünglich genutzten proprietären Produkten in den folgenden Abschnitten beschränkt sich auf solche Programme, welche in der Geschäftsstelle der Monopolkommission häufig genutzt werden. Hierbei handelt es sich um Staroffice[3], IceWM, Sylpheed, Galeon und Gnome Midnight Commander (GMC).

Weiterhin wird erläutert, welche Funktionalitäten im Bereich der Bildbearbeitung sowie der Datenerfassung und -auswertung nicht durch Open-Source-Software ersetzt wurden. Die im folgenden beschriebenen Probleme wurden in der Mehrzahl intensiv mit den Vertretern der mit der Durchführung der Migration beauftragten Unternehmen diskutiert. Ihr Weiterbestehen lässt aus Sicht der Endanwender vermuten, dass sie auch durch EDV-Experten keiner einfachen Lösung zugeführt werden können.

3.1. Büroanwendungen: Staroffice

Staroffice wird in der Geschäftsstelle der Monopolkommission in der Version 6.0 hauptsächlich zur Textverarbeitung und weniger häufig zur Tabellenkalkulation genutzt.

Im Vergleich zu den MS-Office-Programmen benötigt Staroffice sehr viel Zeit für den Programmstart (ca. 40 Sekunden). Zusätzlich wird die Nutzung anderer Programme in diesem Zeitraum vereitelt, da sich während des Programmstarts ein 12 x 10 cm großes Staroffice-Emblem auf allen Arbeitsebenen zentral im Bildschirmvordergrund öffnet. Ein unbeabsichtigtes Beenden von Staroffice kommt recht häufig vor, denn das Programm beendet sich automatisch, sobald kein Dokument mehr geöffnet ist. Der unerfahrene Nutzer bekommt so häufig die Gelegenheit zu einer ausgiebigen Betrachtung des Staroffice-Logos. Allerdings stehen auf Grund seines monolithischen Aufbaus – Staroffice beinhaltet neben einem Textverarbeitungswerkzeug weitere Module zur Tabellenkalkulation sowie zur Erstellung von Präsentationen und Zeichnungen – weitere Anwendungsmöglichkeiten ohne einen erneuten langwierigen Programmstart zur Verfügung.

Insgesamt kann Staroffice nach den Erfahrungen in der Geschäftsstelle der Monopolkommission nicht als grundsätzlich *stabiler* als MS Word betrachtet werden. Einige Mitarbeiter hatten häufige Programmabstürze zu verzeichnen, deren Ursache ungeklärt blieb. Die Eigenheiten von Staroffice, gelegentlich einzelnen Nutzern ein Speichern ihrer Dokumente nicht zu ermöglichen, führte des Öfteren zu einer spannungsreichen Belebung des Arbeitsalltags. Eine Kollegin war am Tag des Veröffentlichungstermins des von ihr bearbeiteten Sondergutachtens von dieser Störung betroffen und musste kurzfristig unter einem anderen Benutzeraccount die vorgesehenen Korrekturen noch einmal eingeben. Das Problem wurde durch die externen EDV-Experten auf die Benutzereinstellungen von Staroffice zurückgeführt und

[3] Es sei darauf hingewiesen, dass Staroffice keine Open-Source-Software ist. Allerdings basiert es auf den Arbeiten der Openoffice-Gemeinschaft.

schließlich behoben, indem die Benutzereinstellungen durch eine ursprüngliche Version ersetzt wurden.

Der *Funktionsumfang* von Staroffice im Bereich der Textverarbeitung ist mit dem von MS Word nahezu vergleichbar. Eine nützliche Neuerung stellt der „Navigator" dar, der die Struktur von Textdokumenten übersichtlich darstellt und es ermöglicht, Überschriften, Tabellen, Grafiken und weitere Textelemente direkt anzusteuern. In der Praxis haben sich jedoch einige Programmfehler bzw. gegenüber MS Word eingeschränkte Funktionen als störend erwiesen, die z.B. die Nutzung von Querverweisen, die Nummerierung von Kapitelüberschriften und die Sprachwerkzeuge betreffen.

Querverweise auf Tabellen und Abbildungen sind in Staroffice 6.0 nicht zu verwenden, da sie mit dem Schließen des Dokuments oder dem Aktualisieren der Felder verloren gehen.

Zur Nummerierung von Überschriften besteht in Staroffice einerseits die Möglichkeit, das Nummerierungsformat einmalig für alle Überschriften festzulegen („Kapitelnummerierung"). Diese Alternative ist mit dem Nachteil einer eingeschränkten Formatierbarkeit der Überschriften behaftet. Andererseits besteht die recht umständliche Möglichkeit, den Überschriften jeweils einzeln ein Nummerierungsformat zuzuweisen.

Die *Sprachwerkzeuge* von Staroffice sind gemessen an seinem proprietären Pendant noch unausgereift. So verfügt das Synonymwörterbuch von Staroffice im Vergleich zu MS Word über einen nur sehr eingeschränkten Wortschatz. Auch funktioniert die Rechtschreibprüfung von Staroffice nur teilweise. Fehler in der Groß- und Kleinschreibung werden gänzlich ignoriert.

Da ein regelmäßiger Austausch von Textdokumenten zwischen den Mitarbeitern der Monopolkommission und den proprietäre Software nutzenden Kommissaren sowie weiteren externen MS-Office-Nutzern erfolgt, wurde die eingeschränkte *Konvertierbarkeit* von Texten zwischen Staroffice und MS Word als die gravierendste mit der Migration verbundene Arbeitserschwernis empfunden.

In der Regel ist eine Veränderbarkeit der Textdateien durch die externen Kooperationspartner erforderlich. Daher stellt die Verwendung des nur lesbaren pdf-Dateiformates für die Arbeitsabläufe in der Geschäftsstelle der Monopolkommission keine Alternative zu einer Konvertierung zwischen Staroffice und MS Word dar. Die in der Geschäftsstelle der Monopolkommission aufgetretenen Konvertierungsprobleme sind äußerst vielfältig und werden im Folgenden nur beispielhaft wiedergegeben:

- Die Kapitelnummerierung aus Staroffice geht bei der Konvertierung nach MS Office verloren.
- Mit Staroffice erstellte Textdateien, in die ein Inhaltsverzeichnis eingefügt wurde, können häufig nach einer Konvertierung nicht mehr durch Staroffice geöffnet werden. Der Versuch, sie zu öffnen, führt zum ungeplanten Beenden des Programms.
- Diagramme, die mit dem Staroffice-Tabellenkalkulationsmodul erstellt wurden, werden in der Regel nach einer Umwandlung in ein Word-Dokument

nicht mehr korrekt wiedergegeben. Gleiches gilt für Diagramme, die mit Excel erstellt und in ein Word-Dokument eingefügt wurden.
- In MS Word zugewiesene Absatznummerierungen gehen bei der Konvertierung nach Staroffice verloren.
- Verschachtelte Tabellen werden durch die Konvertierung verzerrt.

MS Word bietet im Gegensatz zu Staroffice keine Möglichkeit, eine Interoperabilität[4] beider Anwendungen durch das Speichern einer Textdatei im Format der Schwesterapplikation zu ermöglichen. Eine Konvertierung ist daher immer durch Staroffice vorzunehmen.

Ein Vorteil von Staroffice bei der Erstellung von *Seriendokumenten* besteht in der Möglichkeit, sich die Datenquelle über dem Seriendokument anzeigen zu lassen. Hieraus können Datenbankfelder durch Anklicken und Ziehen in das Seriendokument bequem eingefügt werden. Andererseits ist das Einfügen komplexerer Feldbefehle, etwa eines bedingten Textes, dessen Bedingung sich auf ein Datenbankfeld bezieht, in Staroffice wesentlich schwieriger durchzuführen als in MS Word, da MS Word dem Nutzer zu diesem Zweck serienbriefspezifische Auswahlfelder und Dialoge anbietet. Auch können einmal eingefügte Feldfunktionen in Staroffice nicht mehr nachträglich editiert werden, was ihre Anpassungsmöglichkeiten zusätzlich einschränkt.

Das *Tabellenkalkulationsmodul* von Staroffice benötigt verglichen mit Excel sehr viel Zeit zum Laden von Dokumenten, die Diagramme enthalten. Ansonsten sind im bisherigen Gebrauch keine wesentlichen Vor- oder Nachteile der Applikation aufgefallen.

Für die Erstellung von Dokumenten im Adobe-Portable-Document-Format (pdf) wurde auf Basis des Betriebssystems Windows 2000 der Adobe Acrobat in der Version 5.0 eingesetzt. Von den verschiedenen Funktionen wurde im Wesentlichen der Acrobat Distiller benutzt, um aus Word-Dokumenten entsprechende pdf-Dokumente zu erstellen. Als Open-Source-Software-Lösung wird nun zur Erzeugung von pdf-Dokumenten GNU Ghostscript 7.05 als pdf-Konverter über einen Druckbefehl in Staroffice aufgerufen. Vor der Migration wurde hierzu der Adobe Acrobat Writer genutzt. Mit der Umstellung auf GNU Ghostscript/Staroffice sind die folgenden praktischen Probleme verbunden:
- Es werden nur einfache pdf-Dateien erzeugt. Der Anwender hat anders als bei Nutzung des Acrobat Distiller keine Möglichkeit, dialoggesteuert die Erstellung von komplexeren Eigenschaften wie Verknüpfungen oder Inhaltsverzeichnissen herbeizuführen.
- Der Druckaufruf ist umständlich, da die Standardeinstellungen im Druckdialog an drei Stellen verändert werden müssen. Zusätzlich birgt das automatische Vervollständigen des Dateinamens zu dem Namen des Staroffice-Dokuments, inklusive der Staroffice-spezifischen Dateierweiterung „sxw", die Gefahr, dass die Originaldatei durch die pdf-Datei überschrieben wird.

[4] Interoperabilität bezeichnet das Ziel, Unterschiede (z.B. uneinheitliche Protokolle oder Formate), die eine Zusammenarbeit von verschiedenen Systemem verhindern, zu überbrücken.

Für das Öffnen von pdf-Dokumenten zum Lesen und Drucken steht der Acrobat Reader in der Version 5.0 als Freeware auch für Linux zur Verfügung.

In der täglichen Praxis erwies sich als störend, dass einige gängige *Schriftarten*, unter anderem Times und Times New Roman, unter Staroffice in der Bildschirmansicht eine mangelhafte Darstellungsqualität besitzen.

Die *Online-Hilfe* von Staroffice ist deutschsprachig und umfangreich. Neben einem Inhaltsverzeichnis, einem Index und einer Suchfunktion bietet sie die Möglichkeit, einzelne Abschnitte mithilfe von Lesezeichen leichter auffindbar zu machen. Diese Lesezeichen werden allerdings aus ungeklärter Ursache regelmäßig gelöscht.

Mittlerweile wird Staroffice in der Programmversion 7.0 vertrieben. Den veröffentlichten Herstellerangaben zufolge treten einige der beschriebenen Probleme, z.B. die Umständlichkeit der Erzeugung von pdf-Dateien, in der neuen Version nicht mehr auf.

3.2. Fenstermanager: IceWM

Als Fenstermanager unter Linux wird in der Geschäftsstelle der Monopolkommission der IceWM X11 Windowmanager eingesetzt, der somit den MS-Windows-Desktop ersetzt.

Ein Vorteil des Fenstermanagers IceWM gegenüber dem MS-Windows-Desktop stellt die Möglichkeit dar, mehrere *Arbeitsflächen* zu nutzen. Allerdings sollte während des Startens eines Programms nicht die gewünschte Arbeitsfläche verlassen werden, da das Programm ansonsten auf der gerade aktivierten Arbeitsfläche läuft. Auch sind die verschiedenen Arbeitsflächen nur eingeschränkt dafür geeignet, identische Programme mehrfach zu starten. Für die Schriftart, Schriftgröße und Farbgestaltung hält IceWM eine Auswahl unterschiedlicher Einstellungspakete („Themes“) bereit.

Die *Online-Hilfe* des IceWM steht nur in englischer Sprache zur Verfügung und warnt gleich zu Anfang mit dem Hinweis „This document is incomplete. Almost everything is subject to change.“ vor überhöhten Erwartungen.

Die *Personalisierung* der Fensterumgebung ist mit dem IceWM zwar vielseitiger gestaltbar, aber auch weniger einfach und intuitiv zu bewerkstelligen als mit dem MS-Windows-Desktop, da die Konfiguration des IceWM in weiten Teilen das Editieren von Textdateien voraussetzt.

Zum Beispiel ist eine Veränderung des Startmenüs mit dem MS-Windows-Desktop mittels „Drag&Drop“ einfach den persönlichen Bedürfnissen anzupassen. Derselbe Vorgang erfordert bei Nutzung des IceWM die Kenntnis der relevanten Konfigurationsdateien und der korrekten Formulierung der Dateieinträge. Diese Kenntnis konnte erarbeitet werden, indem die Lektüre der Online-Hilfe durch einen längeren Prozess des Versuchs und Irrtums ergänzt wurde.

3.3. E-Mail: Sylpheed

Als Mail-Client wird von der überwiegenden Zahl der Mitarbeiter der Monopolkommission Sylpheed in der Version 0.8.11 genutzt. Vor der Migration wurde zu diesem Zweck Outlook Express 5 verwendet. Mittels eines einfachen Perl-Skripts

konnten die aus Outlook exportierten E-Mail-Adressen in das von Sylpheed benötigte ldif-Format konvertiert und anschließend in das Adressbuch von Sylpheed importiert werden.

Verglichen mit seinem Pendant MS Outlook sowie dem ebenfalls quelltextoffenen Mozilla, wurde als gravierender Nachteil von Sylpheed empfunden, dass es keine Möglichkeit gibt, ein akustisches oder optisches Signal zu erhalten, welches auf den Eingang einer E-Mail hinweist. Ansonsten erfüllt Sylpheed die gegebenen Anforderungen.

3.4. HTML-Browser: Galeon

Als Open-Source-basierter Internetbrowser wird in der Geschäftsstelle der Monopolkommission überwiegend Galeon in der Version 1.2.9 als Ersatz für den Internet Explorer 5 eingesetzt.

Einen Vorteil von Galeon gegenüber dem bislang genutzten MS Internet Explorer stellt die Lesezeichen-Werkzeugleiste dar, welche eine übersichtliche Organisation der Lesezeichen ermöglicht.

Probleme auf Grund der browserspezifischen Gestaltung von Webseiten traten sehr selten auf. Allerdings lässt sich im Gegensatz zum MS Internet Explorer sowohl mit Galeon als auch mit Mozilla häufig nur die erste Seite von mehrseitigen HTML-Dokumenten, für deren Gestaltung Frames verwendet wurden, problemlos ausdrucken. Um einen vollständigen Ausdruck zu erhalten, sind die Markierung des Textes und die Beschränkung des Druckbefehls auf die Selektion oder das Öffnen des Frames in einem separaten Fenster notwendig. Auch öffnet aus ungeklärter Ursache der quelltextoffene Browser in einigen Fällen pdf-Dateien nicht automatisch durch Starten des Acrobat Reader.

3.5. Dateiorganisation und -suche: GNOME Midnight Commander

Der GNOME Midnight Commander (GMC), der den Mitarbeitern der Monopolkommission in der Version 4.5.55 zur Verfügung steht, bietet eine grafische Oberfläche zur Organisation von Dateien und Ordnern. Hinsichtlich der Funktion der Dateisuche ist er als Ersatz für den Windows-Suchassistenten vorgesehen. Für andere Einsatzmöglichkeiten des GMC, etwa die Verschiebung, Vervielfältigung oder Löschung von Dateien, steht unter Windows der Explorer zur Verfügung.

Verglichen mit dem Windows-Suchassistenten, bietet der GMC einem nicht über Spezialkenntnisse verfügenden Anwender einen erheblich geringeren Funktionsumfang zur Dateisuche. Mit dem GMC ist mittels einer einfachen Feldeingabe lediglich eine Suche nach Dateinamen und Dateiinhalt möglich, während der Windows-Suchassistent weitere Suchkriterien, z.B. Datum und Dateigröße, anbietet. Für eine Detailsuche mittels GMC ist die Kenntnis von Shell-Suchmustern oder regulären Ausdrücken[5] nötig. Ein Speichern der Suchergebnisse ist mit dem GMC im Gegensatz zum Windows-Suchassistenten nicht möglich. Zusätzlich sind nach dem Wechsel in ein als Suchergebnis angezeigtes Verzeichnis mittels GMC die restlichen

[5] Ein regulärer Ausdruck ist eine Folge von normalen Textzeichen und/oder Spezialzeichen, wobei Textzeichen für sich selbst stehen und Spezialzeichen Operatoren darstellen, mit deren Hilfe komplexe Textmuster beschrieben werden können.

Suchergebnisse nicht mehr verfügbar. Auch ist eine Navigation in den Verzeichnissen per GMC nicht möglich, solange das Fenster mit den Suchergebnissen geöffnet ist.

Auf den ersten Blick ist für einen weniger routinierten GMC-Nutzer nicht deutlich, in welchen Verzeichnissen die Suche stattfindet. Die Voreinstellung der Suche weist auf das aktuell markierte und angezeigte Verzeichnis. Dies wird jedoch in dem Fenster, in dem die Suchparameter einzugeben sind, nur recht kryptisch durch einen Punkt im Feld „Anfangen bei" angezeigt.

Im Gegensatz zum MS Explorer lassen sich im GMC keine Lesezeichen anlegen. Es existiert lediglich ein Button, mit dem das Heimatverzeichnis angesteuert werden kann. Der Editor für die Verknüpfungen zwischen Dateinamenerweiterungen und Dateitypen (Mimetypes) lässt sich nicht von GMC aus aufrufen, obwohl hierfür ein Menüeintrag existiert.

3.6. Systemmonitor: GTop

Als Systemmonitor wird von einigen Mitarbeitern der GNOME GTop 1.0.13 verwendet. GTop zeigt die Prozessorauslastung, die Speicherbelegung und die Dateisystembelegung an. Von den Mitarbeitern der Monopolkommission wird er hauptsächlich genutzt, um abgestürzte Programme zu beenden.

Für den unerfahrenen Anwender ist verwirrend, dass GTop die einzelnen Teile (Threads) der Programme separat anzeigt. Auch wird für jeden Thread die Speicherbelegung des gesamten Programms wiedergegeben. Nachteilig gegenüber dem Windows-Taskmanager ist, dass GTop nicht über eine voreingestellte Tastenkombination aufgerufen werden kann. Das Beenden eines Programms ist mittels GTop durch das Markieren eines zugehörigen Threads und das Aufrufen eines Signals über das Kontextmenü möglich. Für den Nutzer einfacher auffindbar wäre zu diesem Zweck ein Button oder ein Eintrag im Hauptmenü.

Eine Online-Hilfe zu GTop ist zwar offenbar grundsätzlich vorgesehen, die entsprechenden Dateien sind jedoch nicht vorhanden.

3.7. Applikationen zur Datenerfassung und -auswertung

Für die empirischen Arbeiten in der Geschäftsstelle der Monopolkommission wird eine Programmierumgebung, ein Werkzeug zur Datenbankadministration, ein Werkzeug zur Formulierung von Abfragen und ein Berichtsgenerator benötigt. Von zwei wissenschaftlichen Mitarbeitern der Monopolkommission, zu deren Arbeitsgebiet die Erhebung und Auswertung empirischen Datenmaterials gehören, wird MS Access unter Windows 2000 zur Eingabe und Verarbeitung der Daten verwendet. Ein Umstieg auf Open-Source-Software ist aus den folgenden Gründen in diesem Bereich bisher nicht erfolgt:

- Für ökonometrische und konzentrationsstatistische Analysen und Datenbankrecherchen wurden umfangreiche und komplexe Prozeduren auf der Basis von *Visual Basic for Application* (MS VBA) zur Anwendung unter MS Access, MS Excel und MS Word entwickelt. Diese sind nur mit erheblichem Aufwand unter Linux nachzubilden.

- Die alternative Verwendung von PostgreSQL wurde erprobt, jedoch in Ermangelung einer ausgereiften kostenfreien grafischen Datenbankschnittstelle verworfen. Als Datenbankinterface wurden PgAccess in der Version 0.98.7, Staroffice 6.0 sowie phpPgAdmin (2.4-1) einer intensiven Prüfung unterzogen. Alle drei Alternativen wiesen jedoch nicht den erforderlichen Funktionsumfang auf.

3.8. Bildbearbeitung

Ein spezielles Problem, das nach der Migration auftrat und noch keiner zufrieden stellenden Lösung zugeführt wurde, stellt der Umgang mit mehrseitigen Dateien im Tagged Image File Format (tif) dar. Den Mitarbeitern der Monopolkommission werden des Öfteren Dokumente ausschließlich im tif-Dateiformat zur Verfügung gestellt. Mit Imaging für Windows war es ursprünglich auf sehr einfache Weise möglich, mehrseitige tif-Dateien auszudrucken. Als Open-Source-basierte Anwendungen wurden zu diesem Zweck GIMP (GNU Image Manipulation Program) und Image Magick getestet. Es ist jedoch nur mit Image Magick und hier nur mit erheblichem Zeitaufwand möglich, den Druck vorzunehmen, da für jede einzelne Seite der Druckbefehl aufgerufen und die Druckereinstellungen modifiziert werden müssen. Als Abhilfe wurde dazu übergegangen, die benötigten tif-Dateien von Windows-Nutzern ausdrucken zu lassen.

3.9. Nutzung mobiler Datenträger

Auf Grund der unterschiedlichen Dateisystemtypen gestaltet sich der Austausch von Dateien mit Windows-Nutzern über mobile Datenträger schwierig. Zwar ist ein Zugriff auf Disketten mit dem Windows-typischen Dateisystemtyp vfat grundsätzlich durch die Angabe der entsprechenden Option möglich. Jedoch besteht mittels der dem Diskettenzugriff durch die Endanwender dienenden Verknüpfung keine Möglichkeit, diese Optionsvorgabe vorzunehmen.

Die Nutzung von CD-ROM ist für die ausschließlich Linux-basierte Software nutzenden Mitarbeiter der Monopolkommission nur eingeschränkt möglich, sofern der Datenzugriff über mitgelieferte Windows-basierte Programme erfolgt. Für einige ausgewählte Datenträger wurde ein Zugriff mittels der Windows-Emulation Wine in der Version 20020411 realisiert.

3.10. Gesamteindruck

Insgesamt besteht der Eindruck, dass die gesamte EDV nach der Umstellung auf Open-Source-Software langsamer arbeitet. Der Start der häufig verwendeten Open-Source-Programme dauert nahezu ohne Ausnahme länger als der Start ihrer proprietären Pendants. Auch das Laden von Dateien nimmt längere Zeit in Anspruch. So kommt es nach der Migration häufiger dazu, dass Nutzer, nachdem sie Aktionen mit hoher Ressourcenbelastung in Gang gesetzt haben, unter dem falschen Eindruck, der Rechner sei inaktiv, durch weitere Befehle und Mausklicks einen Rechnerabsturz herbeiführen.

Eine Windows-freie Zone ist nach der Migration nicht entstanden. Zum einen existieren noch Windows-Enklaven im Bereich der empirisch orientierten Tätigkei-

ten. Zum anderen sind die Open-Source-Anwender unter den Mitarbeitern der Monopolkommission des Öfteren – etwa bei der Prüfung von Konvertierungsergebnissen und dem Drucken mehrseitiger tif-Dateien – auf die Unterstützung von Windows-Nutzern angewiesen.

Grundsätzlich stehen dem Nutzer eine sehr große Zahl quelltextoffener, kostenloser Applikationen zur Auswahl. In der praktischen Anwendung erweist sich jedoch häufig, dass diese zwar die grundlegenden Funktionen ihrer proprietären Pendants besitzen, jedoch erhebliche Mängel hinsichtlich fortgeschrittener Funktionen, Benutzerkomfort und Dokumentation aufweisen. Sie sind in der Regel weniger selbsterklärend und nicht intuitiv nutzbar und besitzen häufig keine bzw. eine wenig umfangreiche oder fremdsprachliche Hilfe.

4. Fazit und Empfehlungen

Ohne Zweifel ist als ein erheblicher Vorteil der Migration aus Nutzerperspektive die Immunität gegen Windows-Würmer und Viren anzusehen. Auch gehören Störungen auf Seiten des File-Servers der Vergangenheit an. Als weitere Verbesserungen des Nutzungskomforts sind die Möglichkeit der Verwendung mehrerer Arbeitsebenen unter IceWM sowie die Verfügbarkeit von im Zuge der Migration erstellter, jedoch nicht Open-Source-Software-spezifischer Applikationen, z.B. der elektronische Bibliothekskatalog, zu nennen.

Die Nachteile der Migration liegen im Wesentlichen in

- den Kompatibilitätsproblemen, die im täglichen Austausch mit Nutzern Windows-kompatibler Software bestehen,
- dem Fehlen einzelner Funktionen, etwa einer ausgereiften grafischen Datenbankschnittstelle,
- der zum Teil geringeren Bedienungsfreundlichkeit der Anwendungen, die hier exemplarisch in den Bereichen Seriendruck, Fenstermanager, Dateisuche und Systemmonitor dargestellt wurde,
- der Notwendigkeit, selbst erstellte, auf proprietärer Software basierende Makros nachzubilden,
- einem erhöhten Bedarf an Unterstützung durch externe Berater.

Unter Wirtschaftlichkeitsgesichtspunkten sind somit neben den Kosten der Migration die Kosten des fortlaufenden Anwendersupports und eines erhöhten Arbeitsaufwandes auf Seiten der Endanwender zu nennen. Die Kosten des Anwendersupports durch externe Dienstleister sind in der Anfangsphase der Open-Source-Nutzung auf Grund der Unerfahrenheit der Anwender und Administratoren sowie längerfristig auf Grund der eingeschränkten Bedienungsfreundlichkeit einiger Programme vergleichsweise hoch. Gleichzeitig verursachen die Qualitätsmängel der Anwendungsprogramme auf Seiten der Endanwender einen erhöhten Zeitbedarf für die Analyse und Kommunikation der auftretenden Probleme und die Anwendung der von EDV-Experten empfohlenen Umgehungstechniken. Auch führen die mit der weiten Verbreitung Windows-basierter Software verbundenen Netzeffekte zu erhöhten Kosten der Nutzung von Open-Source-Software, etwa in Form des Arbeitsaufwandes, der mit der Konvertierung der Dateiformate, der Überprüfung der

Konvertierungsergebnisse und der Analyse und Behebung von Umwandlungsproblemen verbunden ist.

Die im Falle der hier beschriebenen Migration verbliebene Notwendigkeit, Windows-Enklaven zu belassen, um selbst erstellte Applikationen weiterhin zu nutzen, wäre möglicherweise durch eine stärkere Berücksichtigung der Problematik im Rahmen der Projektkonzeption vermeidbar gewesen.

Die Rolle der Nutzer war im vorliegenden Projekt weitgehend auf das Testen der installierten Software, die Äußerung von Detailwünschen sowie die Berichterstattung über Probleme reduziert. Grundsätzlich erscheint zur Erzielung einer höheren Produktivität und Nutzerzufriedenheit eine stärkere Einbeziehung der von der Migration betroffenen Arbeitnehmer in die Projektplanung und -erfolgskontrolle wünschenswert. Hinsichtlich der Schulungen ergibt sich im Nachhinein der Eindruck, dass eine zusätzliche bzw. ausführlichere Einführung aller Mitarbeiter in die Nutzung der Applikationen zur Dateiorganisation und -suche sowie des Fenstermanagers zielführend gewesen wäre.

Bei Abwägung der Frage, ob ein Umstieg auf Open-Source-Software sinnvoll ist, ist neben einer Kosten- und Sicherheitsbetrachtung eine Beurteilung hinsichtlich der Arbeitsproduktivität sinnvoll. Als *Entscheidungsgrundlage hinsichtlich einer Client-Migration* sollten daher ergänzend die folgenden Kriterien herangezogen werden:

- die digitalen Schnittstellen zur Außenwelt,
- der Abdeckungsgrad, mit dem ursprünglich genutzte Funktionalitäten durch Open-Source-Software ersetzt werden können,
- die Technikorientierung der Anwender.

Bezüglich der digitalen Schnittstellen des betrachteten Arbeitsbereichs zu Nutzern anderer Betriebssysteme und Applikationen ist zu untersuchen, in welchem Umfang und mittels welcher Medien und Dateiformate ein Informationsaustausch erfolgt. Ein Übermittlungsmedium mit geringem Problempotenzial stellt das Internet dar, während die Nutzung mobiler Datenträger durch die differierenden Dateisystemtypen nicht ohne weiteres funktioniert. Auf Grund der beschriebenen Konvertierungsprobleme und Netzeffekte ist eine geringere Nutzerzufriedenheit und -produktivität zu erwarten, sofern per MS Office erzeugte Dateiformate für den Informationsaustausch verwendet werden müssen. Weniger problematisch ist hingegen das Dateiformat pdf. Offene Dateiformate wie der American Standard Code for Information Interchange (ascii), txt oder html verursachen zwar keine Konvertierungsprobleme, sind jedoch im Büroalltag auch weniger gebräuchlich und von geringerem Nutzwert.

Als Entscheidungsgrundlage ist weiterhin eine detaillierte Analyse der EDV-gestützten Arbeitsabläufe der betroffenen Mitarbeiter zielführend, um festzustellen, in welcher Intensität die einzelnen Funktionen der zu ersetzenden Applikationen genutzt werden. In einem zweiten Schritt sollte geprüft werden, ob die häufig genutzten Funktionen durch die Open-Source-Anwendungen in ausreichender Qualität zur Verfügung gestellt werden. Die pauschale Einschätzung des Ersatzbedarfs auf Ebene der gesamten Applikationen ist hingegen auf Grund des unterschiedlichen

Funktionsabdeckungsgrades der in einem Austauschverhältnis stehenden Programme nicht ausreichend.

Schließlich erweist sich angesichts der zum Teil noch geringen Ausrichtung der Applikationen an der Wahrnehmung und den EDV-Kenntnissen von „Normalnutzern" und der zu erwartenden Abhängigkeit von externem Support die Technikorientierung der Endanwender als wichtig für den Migrationserfolg. Erfahrungsgemäß stellt sie einen entscheidenden Faktor für die Möglichkeiten der Endanwender dar, auftretende Probleme selbst zu beheben oder angemessen zu kommunizieren.

Eckdaten zur Monopolkommission:

Die Monopolkommission ist ein durch das Zweite Gesetz zur Änderung des Gesetzes gegen Wettbewerbsbeschränkungen vom 3.8.1978 gebildetes Sachverständigengremium. Die Mitglieder werden auf Vorschlag der Bundesregierung durch den Bundespräsidenten auf Dauer von vier Jahren berufen. Sie sind in ihrer Tätigkeit unabhängig und nur an ihren gesetzlichen Auftrag gebunden. Gesetzlicher Auftrag der Monopolkommission sind die Beurteilung des Stands der Unternehmenskonzentration in der Bundesrepublik Deutschland sowie deren absehbare Entwicklung unter wirtschafts- , insbesondere wettbewerbspolitischen Gesichtspunkten und die Würdigung der Entscheidungspraxis der Kartellbehörden und der Gerichte zur Missbrauchsaufsicht und zur Fusionskontrolle. Darüber hinaus ist die Kommission aufgefordert, nach ihrer Auffassung notwendige Änderungen der einschlägigen Bestimmungen des Gesetzes gegen Wettbewerbsbeschränkungen aufzuzeigen. In Durchführung dieses gesetzlichen Auftrags hat die Monopolkommission alle zwei Jahre zum 30. Juni ein Gutachten zu erstellen, das der Bundesregierung zugeleitet wird, die es den gesetzgebenden Körperschaften vorlegt und in angemessener Frist dazu Stellung nimmt. Die Untersuchungsergebnisse werden damit zum Gegenstand parlamentarischer Diskussion und durch die vorgeschriebene Veröffentlichung darüber hinaus einer breiten Öffentlichkeit bekannt. Über die regelmäßige Beurteilung der Konzentrationsentwicklung hinaus erstattet die Monopolkommission zusätzliche Gutachten (sog. „Sondergutachten") sowohl im Auftrag der Bundesregierung als auch nach eigenem Ermessen. Darüber hinaus hat der Bundesminister für Wirtschaft seit der Vierten GWB-Novelle 1980 in allen Zusammenschlussfällen, in denen er im Rahmen eines sog. Ministererlaubnisverfahrens zu entscheiden hat, die gutachterliche Stellungnahme der Monopolkommission einzuholen.

Die Geschäftsstelle der Monopolkommission befindet sich in Bonn. Derzeitige Mitglieder der Monopolkommission sind:

Martin Hellwig (Vorsitzender, Mannheim), Jörn Aldag (Hamburg), Jürgen Basedow (Hamburg), Katharina M. Trebitsch (Hamburg).

Migration der Server- und Desktoplandschaft im Landesrechnungshof Mecklenburg-Vorpommern

FRANK MÜLLER

Der Landesrechnungshof Mecklenburg-Vorpommern ist eine oberste Landesbehörde und nur dem Gesetz unterworfen. Seine Mitglieder genießen richterliche Unabhängigkeit. Diese Unabhängigkeit betrifft auch den Bereich Informationstechnik. Im Landesrechnungshof arbeiten 100 Mitarbeiter an zwei Standorten, Neubrandenburg und Schwerin. Hauptanwendungen sind Produkte aus dem Office-Bereich (MS Office 95, MS Office 2000, Visio 2000), Groupware-Anwendungen aus dem Microsoft-Exchange-5.5-Umfeld und verschiedene, kleinere Fachanwendungen im Datenbankumfeld des Microsoft-SQL-Servers 7.0. Die Ziele der Umstellung lagen einmal in der Unterstützung der Prüfer bei der:

- Informationsbeschaffung
- Verarbeitung der Daten
- internen und externen Kommunikation
- Ablage und Archivierung der Prüfungsergebnisse
- jährlichen Veröffentlichung der Prüfungsergebnisse im Jahresbericht und außerdem in einer stabilen, sicheren und kostengünstigen IT-Umgebung.

Ausgehend von dem eingestellten Support von Microsoft für Windows NT 4.0, einer sehr uneinheitlichen Situation im Office-Bereich und der neuen Lizenzpolitik der Firma Microsoft, stand das IT-Referat vor der Aufgabe, mögliche Wege aus diesem Dilemma zu suchen, an deren Zielpunkt eine wirtschaftliche, lizenzrechtlich saubere und möglichst einheitliche Lösung stehen sollte. Auch im Hause wuchs die Unzufriedenheit mit der augenblicklichen Situation, in vielen Mails fanden die Anwender Dokumente der verschiedensten Formate von Microsoft Office, die oftmals durch das IT-Referat umformatiert werden mussten. In diesem Zusammenhang sei angemerkt, dass die Einführung von Standards sehr hilfreich sein kann. Dokumentenaustausch sollte grundsätzlich auf Basis des Rich-Text-Formats[1] (wenn es weiterbearbeitet werden soll) oder mittels Adobes-Portable-Document-Format[2] erfolgen. Auch bei den Administratoren wuchs die Unzufriedenheit, da mittlerweile drei Betriebssysteme zu unterstützen waren und mehr Zeit für die Suche für diverse Treiber aufgewendet werden musste als für die anderen notwendigen Arbeiten. Es galt, sich nach neuen Wegen umzusehen, um nicht ständig an der wachsenden IT-Patchworkdecke herumzuflicken.

Das Spektrum der möglichen Lösungsansätze reichte von einer reinen Microsoft-basierten Lösung für alle Komponenten im Haus (sowohl Server als auch

[1] Rich-Text-Format-Dateien mit der Endung rtf, durch jede Textverarbeitung zu erstellen und weiterzubearbeiten.

[2] Portable-Document-Format-Dateien mit der Endung pdf, pdf-Dokumente können unter allen Betriebssystemen mit dem Acrobat Reader gelesen werden.

Desktops) bis hin zur kompletten Linux/Open-Source-basierten Lösung. Dabei wurden insgesamt folgende Teilbereiche betrachtet:

- Serverseitige Software-Kosten,
- Clientseitige Software-Kosten,
- Kosten der Hardware,
- Schulungskosten,
- Kosten für die Migration der Fachanwendung Kommunaldatenbank.

Die bislang im Hause erreichte Funktionalität der Anwendungen sollte dabei für die Anwender erhalten bleiben. Schließlich will man sich ja unter dem Strich in seinen Möglichkeiten verbessern und nicht noch zusätzlichen Einschränkungen unterliegen. Die unbedingt notwendige Akzeptanz für eine neue Lösung ist mit verminderter Funktionalität bei den Nutzern jedenfalls nicht zu erreichen. Es kann eben nur gemeinsam mit den Nutzern und der Personalvertretung passieren. Unter Berücksichtigung dieser Rahmenbedingungen wurden die möglichen Wege einer Migration betrachtet. Allein im Bereich der Kosten traten dabei Differenzen von über 93000 € zwischen der reinen Migrationslösung zu Microsoft und der Linux/Open-Source-Lösung auf. Im Oktober 2002 (Tabelle 1 fasst die gesamte Zeitschiene der Migration zusammen) entschied der Präsident des Landesrechnungshofes, dass im Jahr 2003 eine Migration des ganzen Hauses auf Basis von Linux/Open-Source-Produkten erfolgen soll. Die Entscheidung sorgte für ein zweigeteiltes Echo. Innerhalb des Hauses waren die Bedenken und die Skepsis der Anwender sehr groß, hingegen gab es außerhalb des Hauses fast durchweg positive Meinungen zu dem Vorhaben.

März 2002	Entscheidungsvorbereitung durch das IT-Referat
Oktober 2002	Entscheidung zur Migration nach Linux/OSS durch den Präsidenten
Oktober 2002 bis Mai 2003	verschiedene Piloten und Testinstallationen sowie Lehrgänge der Administratoren
Juni 2003 bis September 2003	System- und Schulungsvorbereitung, Informationsveranstaltungen im Hause
Oktober 2003	Migration der Server
November 2003	Dezember 2003 Migration der Clients und parallele Schulung der Anwender
Januar 2004	Produktivbetrieb der gesamten IT auf Basis Linux/OSS

Tabelle 1: Zeitschiene der Migration

Das IT-Referat bereitete danach verschiedene kleine Pilotprojekte vor, um geeignete Produkte für den zukünftigen Einsatz im Landesrechnungshof zu finden. Gleichzeitig wurde versucht, auf Ausstellungen, Messen und Fachtagungen Kontakte zu knüpfen, um auch an weiter gehende Informationen zu gelangen. Auch für die Administratoren bedeuteten die Rückkehr zur Konsole und damit das vermehrte

Arbeiten mit Befehlen an der Tatstatur und das Fehlen der Maus zunächst einen erhöhten Lernaufwand, waren doch die Kenntnisse aus den Microsoft-Qualifikationen (MCSE – Microsoft Certified Systems Engineer, MCP – Microsoft Certified Professional) jetzt nur noch begrenzt einsetzbar. Schließlich kann man schlecht sagen: Migration? Ja, toll, machen wir. Grundlegende Kenntnisse im Aufbau und der Arbeit unter und mit Linux sollte man haben.

Dazu ist ein einigermaßen grober Marktüberblick notwendig. Aufbauend auf das neue Wissen aus den ersten Weiterbildungsmaßnahmen für die Administratoren, konnte jetzt die Suche nach geeigneten Produkten und Lösungen weiter forciert werden. Dabei waren natürlich insbesondere Lösungen aus dem Verwaltungsumfeld, also bei Bundes- oder Landesbehörden, von speziellem Interesse für uns. Ein entscheidender Vorteil für unser Haus bei der Migration war dabei, dass der Landesrechnungshof autark arbeitet, die Anzahl der Fachanwendungen begrenzt ist und die Anschlüsse an externe Anwendungen strikt getrennt vom hausinternen Netzwerk gehandhabt werden. Das zentrale Arbeitsmittel für die Prüfer ist die Textverarbeitung im Office-Programm. Hier wurde der erste Schwerpunkt für die Migration gesetzt. Verschiedene Testszenarien mit typischen Dokumenten wurden aufgebaut. Schwerpunkte lagen dabei in der Qualität der Import- und Export-Funktionen für Microsoft-Office-Dokumente, der Verarbeitung von Formularen, im Umgang mit großen Textdateien von bis zu 350 Seiten und weiteren spezifischen Anforderungen. Als mögliche Alternativen stellten sich hier Staroffice bzw. Openoffice heraus. Nachteilig für Staroffice war die Lizenzpolitik, da im Modell von Sun keinerlei Updates vorgesehen waren. Trotz der gegenüber der kommerziellen Version etwas eingeschränkten Funktionsvielfalt hat sich dann letztlich Openoffice durchgesetzt.

Damit war die Hauptanwendung geklärt, und die Administratoren konnten sich der Auswahl des Desktopbetriebssystems zuwenden. Dazu wurde von allen vorhandenen Clients jeweils ein Rechner als Versuchsobjekt genutzt. Insgesamt mussten nur drei Modelle getestet werden, die Notebooks wurden hierbei noch außen vor gelassen. Da zwei der Modelle noch mit Intel-Pentium-II-Prozessoren mit 466 MHz bzw. Intel-Celeron-Prozessoren mit 500 MHz ausgestattet sind, wurde auf IceWM als Windowmanager gesetzt. Als Distribution (integrierte Linux-Programmsysteme, bereits fix und fertig) kamen Redhat (7.3 und 8.0), SuSE (8.0 bis 8.2), Mandrake (8.1) und Debian (3.0) in den Test. Aus Zeit und anderen Gründen haben wir davon abgesehen, uns ein eigenes Linux zu „bauen“, obwohl das durchaus ohne weiteres möglich wäre, da Linux zur freien Nutzung im Internet zur Verfügung steht. Ganz ohne Probleme ging es aber bei den „fertigen“ Systemen auch nicht, und die Administratoren fragten sich bisweilen, ob die Skepsis der Anwender doch nicht so unbegründet war. In die zweite Testrunde schafften es Redhat und Debian. Nach weiteren Tests überzeugte die Stabilität von Debian zusammen mit dem Update-Mechanismus.

Danach musste nur noch das Desktop-System für die Anwender vervollständigt werden. Das beinhaltete neben der Office-Komponente auch den Acrobat Reader, verschiedene Tools und insbesondere den E-Mail-Client. Auch hier standen verschiedene Programme zur Verfügung. Neben Sylpheed und Aethera kam ebenso Evolution in Betracht. Die visuelle Nähe zu Outlook gab letztlich für Evolution den

Ausschlag. Wir wollten die Migration den Anwendern natürlich so leicht wie möglich machen. Das „Look&Feel" entsprach ziemlich genau dem von Outlook.

Bereits im Vorfeld haben wir den Nutzern CDs mit Openoffice für den Gebrauch zu Hause angeboten. So wird die erste Angstschwelle vor den neuen Programmen gesenkt, und da Openoffice wie Staroffice auch sowohl unter Windows als auch unter Linux funktionieren, musste lediglich die Office-Suite installiert werden. Zusätzlich haben wir auch die aktuelle Knoppix-CD für den „Heimgebrauch" angeboten. Bei der Knoppix-CD bootet der Rechner ein lauffähiges Linux von der CD einschließlich Office und einer Vielzahl anderer Programme. Beachtlich dabei ist, dass keinerlei Änderungen am bestehenden System vorgenommen werden. Mit diesen „vertrauensbildenden Maßnahmen" hofften wir unsere kritischen und anspruchsvollen Nutzer weiter überzeugen zu können. Niemand lässt sich gern auf etwas Neues, etwas Unbekanntes ein, da musste durch die Administratoren bereits im Vorfeld schon viel argumentiert und auch vorgeführt werden.

Eine weitere große Herausforderung stellt die Migration der Groupware-Lösung dar. Groupware bedeutet eben mehr als eine reine E-Mail-Lösung. Das wäre kein Problem gewesen, viele stabile Mailserver gibt es unter Linux, aber wir wollten eben etwas mehr. Die Suche nach der „Eier legenden Wollmilchsau" ist noch nicht endgültig abgeschlossen, wir werden dazu wohl einen Produktmix einführen, der die vielfältigen Aufgaben der Groupware insgesamt ersetzen soll.

Eine große Hilfe war für uns die EU-Studie zur Migration[3], die wir als Praxisbeispiel begleiten durften. Hier hatten wir kompetente Partner, die unsere Vorstellungen und Pläne zur Migration kritisch begleiteten und dabei gleichzeitig auf Machbarkeit für die Allgemeinheit überprüften. Sicherlich gab es auch unterschiedliche Auffassungen, die aber in der Regel auf den besonderen Anforderungen unseres Hauses in puncto Sicherheit und Unabhängigkeit basieren.

Eine weitere Hilfe fanden wir bei der Firma natural.computing, deren Lösung bei der Bundesmonopolkommission unsere Erwartungen an ein einfach zu verwaltendes Linux-basiertes Netzwerk in fast idealer Weise erfüllte. Nach einem Besuch dieser Behörde und der zeitgleichen Beurteilung verschiedener anderer Lösungen haben sich die Administratoren für die Monopolkommissionslösung entschieden. Sie bietet zudem noch weitere, für den Rechnungshof sehr wichtige Ausbaustufen, nämlich die Nutzung von Biometrie-Daten zur Authentifizierung. Damit entfällt der für die Nutzer so lästige Anmeldevorgang am System mittels Passwort, das Passwort tragen sie ja immer bei sich. Kein Vergessen, kein Wechsel des Passwortes nach einer bestimmten Zeitspanne mehr. Das bedeutet sowohl Erleichterungen für die Nutzer als auch für die Administratoren. Hierbei muss angemerkt werden, dass die Lösung keinerlei biometrische Daten wie Fingerabdrücke speichert, sondern nur mathematische Muster, die selbst wiederum absolut keinen Rückschluss auf den Fingerabdruck zulassen. Wer möchte schon seine biometrischen Daten irgendwo im System, und sei es noch so sicher, hinterlegt haben. Das ist aber noch Zukunftsmusik und wird erst im ersten Halbjahr 2004, nach der Erstellung des Jahresberichts, erfolgen.

[3] Vgl. http://europa.eu.int/ISPO/ida/jsps/index.jsp?fuseAction=home.

Nach der Erstellung eines Pilot-Clients, der alle notwendigen Programme und Einstellungen für die Nutzer enthielt, wurde derselbe einem intensiven Test unterzogen. Da die Lösung der Monopolkommission nicht 1:1 umsetzbar war, bekamen nicht nur die Mitarbeiter bei natural große Ohren vom ständigen Telefonieren mit unseren Administratoren, sondern auch die IT-Mitarbeiter im Rechnungshof litten unter dieser spontan aufgetretenen Mutation. Schließlich wurde es im Spätherbst ernst. Innerhalb von fünf Wochen wurden jeweils in Gruppen zu 20 Mitarbeiter die Rechner umgestellt, und die betroffenen Mitarbeiter geschult. Der erste Teil der Woche wurde umgestellt und am Donnerstag und Freitag hieß es dann jeweils Schulung im Hause. Dazu wurde aus älterer Technik ein Schulungskabinett aufgebaut, um direkt an den neuen Oberflächen und Programmen die auftretenden Fragen und Probleme diskutieren zu können. Die Schulung wurde durch einen entsprechend qualifizierten IT-Mitarbeiter durchgeführt. Dabei wurden besondere Schwerpunkte in Hinsicht auf die Bedienung der neuen Oberfläche, des Mail-Programms und die Unterschiede zwischen dem Microsoft-basierten Office und Openoffice eingegangen. Von wenigen, kleineren Ausreißern abgesehen, verlief die Migration auf der Anwenderseite relativ problemfrei. Dabei konnten die Administratoren einerseits auf viel Verständnis bei den Anwendern bauen, andererseits haben die Administratoren versucht, kleinere Probleme auf der Anwenderseite schnell und unkompliziert zu lösen. Es herrschte also auf beiden Seiten ein großes Verständnis für die Sorgen und Nöte des jeweils anderen. Das ist umso beachtenswerter, da sich das ganze Haus in der Vorbereitung für den Jahresbericht befand, Termindruck also auf allen Ebenen herrschte. Trotzdem funktionierte die Umsetzung der jeweiligen Texte sehr gut, was am Montagmorgen noch ein Jahresberichtsbeitrag in MS Word 2000 oder MS Word 95 war, wurde am Nachmittag bereits mit Openoffice weiterbearbeitet.

Zusammenfassend kann man sagen, dass es für alle eine ungeheure Herausforderung darstellt, eine gesamte Behörde im Produktionsbetrieb zu migrieren. Es erfordert viel Vorbereitung und Verständnis von allen Seiten. Vertrauensbildende Maßnahmen fördern die Akzeptanz schon im Vorfeld und senken merklich die Hemmschwelle bei den Nutzern. Ein erster Erfolg war, dass einige „Poweruser“, wie beispielsweise die Sekretärinnen und einige andere Vielschreiber, uns einige Zeit später anriefen und sagten, dass es wirklich stabiler und schneller geht. Und nach einiger Zeit der Eingewöhnung kann man die allgemeinen Aufgaben genauso gut oder zum Teil sogar noch besser lösen als vorher. Ich will natürlich nicht verschweigen, dass es immer noch einige hartnäckige Kritiker gibt, die Mehrzahl der Nutzer aber hat sich von der Entscheidung zur Migration überzeugen lassen und damit auch den Erfolg dieser Migration gezeigt. Darauf lässt sich doch aufbauen, oder?

Eckdaten zum Landesrechnungshof Mecklenburg-Vorpommern:

Der Landesrechnungshof hat den Verfassungsauftrag, die gesamte Haushalts- und Wirtschaftsführung des Landes einschließlich der übrigen landesunmittelbaren juristischen Personen des öffentlichen Rechts sowie überörtlich die Haushaltsführung der kommunalen Körperschaften zu überwachen.

Im Rahmen seiner Überwachung prüft der Landesrechnungshof auch die Betätigung der öffentlichen Hand in privatrechtlichen Unternehmen. Darüber hinaus ist der Landesrechnungshof zuständig, soweit Stellen außerhalb der Landesverwaltung Landesmittel erhalten oder Landesvermögen verwalten. Wenn juristische Personen des privaten Rechts Mittel aus dem Landeshaushalt erhalten, Landesvermögen verwalten oder dem Landesrechnungshof ein Prüfungsrecht eingeräumt ist, prüft der Landesrechnungshof die Haushalts- und Wirtschaftsführung auch dieser Institutionen.

Schließlich hat der Landesrechnungshof die Aufgaben, den Landtag und die Landesregierung im Rahmen seiner gesetzlichen Aufgaben zu beraten und gutachtliche Stellungnahmen abzugeben. Vor dem Erlass bestimmter Vorschriften muss der Landesrechnungshof gehört werden.

Der Landesrechnungshof ist eine oberste Landesbehörde und als unabhängiges Organ der Finanzkontrolle nur dem Gesetz unterworfen. Seine Mitglieder bilden zugleich das Beschlussorgan Senat. Der Präsident und der Vizepräsident werden vom Landtag gewählt und vom Ministerpräsidenten ernannt. Die weiteren Mitglieder ernennt der Ministerpräsident auf Vorschlag des Präsidenten des Landesrechnungshofes. Der Senat besteht einschließlich Präsident und Vizepräsident aus sechs Mitgliedern, die durch richterliche Unabhängigkeit geschützt sind.

Insgesamt hat der Landesrechnungshof 101 Stellen. Der Landesrechnungshof hat Dienststellen in Neubrandenburg und Schwerin.

Staatsrechtliche Grundlagen:

Neben den bundesrechtlichen Vorschriften (u.a. Haushaltsgrundsätzegesetz HGrG, Beamtenrechtsrahmengesetz BRRG) gelten insbesondere:

- Art. 68 der Verfassung des Landes Mecklenburg-Vorpommern vom 23. Mai 1993 (GVOBl. M-V 1993, Seite 372)
- Rechnungshofgesetz (LRHG) vom 21. November 1991(GVOBl. M-V 1991, Seite 438)
- Landeshaushaltsordnung – Teil V – Bekanntmachung der Neufassung vom 15. Februar 1994 (GVOBl. M-V 1994, Seite 186)

Der Landesrechnungshof hat seinen Dienstsitz in Neubrandenburg. Eine weitere Dienststelle befindet sich in Schwerin. In beiden Dienststellen sind circa 100 Mitarbeiter beschäftigt.

Die Internetseite des Landesrechnungshofes findet sich unter www.lrh-mv.de.

Die KDE-Entwicklergemeinde – wer ist das?

EVA BRUCHERSEIFER

1. Einleitung

KDE, das steht vorläufig noch für „K Desktop Environment". Es ist der Name eines Projekts der Open-Source-Bewegung. Die Entwicklung findet im Internet, statt und so sind „neue Medien" die hauptsächlichen Kommunikationswege der Projektmitglieder. Das Projekt präsentiert sich selbst unter der KDE-Webseite[1].

Im Namen steht „Desktop" dafür, dass KDE eine Open-Source-Plattform für grafische Desktopanwendungen unter GNU/Linux und anderen Unix-Systemen darstellt. In seiner aktuellen Version ist KDE 3.2 der populärste Desktop für Linux und Unix und präsentiert sich dem Anwender mit vielen Features. Mehr als 50 Anwendungen sorgen dafür, dass der Anwender seine Aufgaben erledigen kann: von Korrespondenzpflege und Internetrecherche über das Erstellen von Dokumenten am Computer bis hin zum Erstellen von CDs. Die populärsten Anwendungen sind „Konqueror", der Webbrowser und Dateimanager, und „KMail", das E-Mail-Programm. KMail kann in seiner neuesten Version zusammen mit dem Kalender „KOrganizer" und anderen Modulen zu dem Kommunikationsprogramm „Kontakt" integriert werden (ähnlich Outlook), wobei die Komponenten auch einzeln benutzbar bleiben. Abbildung 1 zeigt den Desktop in seiner aktuellen Form.

[1] Vgl. http://www.kde.org, deutsche Webseite unter http://www.kde.de.

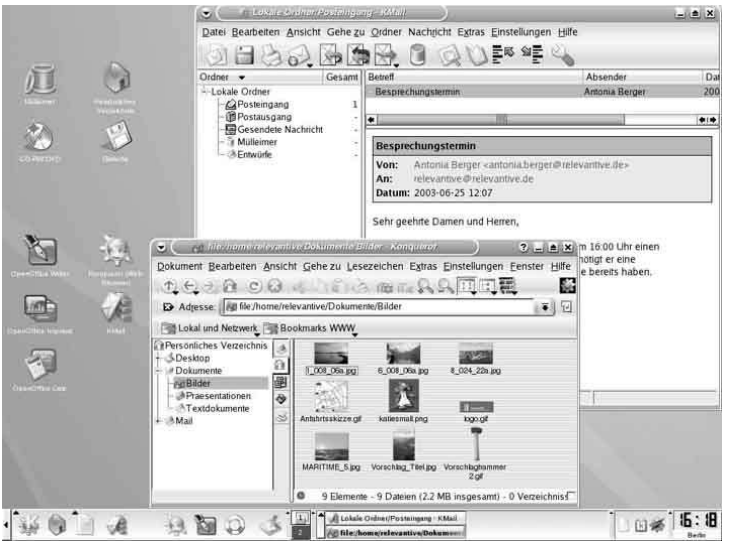


Abbildung 1: Der KDE-Desktop

Das Namenselement „Environment" will aussagen, dass es sich bei KDE um mehr als eine lose Ansammlung von Programmen handelt. Vielmehr stellt KDE eine integrierte Arbeitsumgebung dar. Anwender des Desktops profitieren besonders bei typischen Aufgaben im Heimanwenderbereich ebenso wie in Office-Umgebungen von einer verzahnten Infrastruktur und dem konsistenten und geläufigen „Look&Feel". Eine zentrale Konfiguration und standardisierte Menüs und Dialoge, Tastenkombinationen ermöglichen eine einfache Benutzung. Alle KDE-Anwendungen weisen eine durchgängige Unterstützung für Übersetzungen auf. Die deutsche Sprache ist als Übersetzung sogar komplett verfügbar. KDE bietet Softwareentwicklern eine durchdachte, stabile Plattform mit innovativen Technologien. Mit diesen Features und einer beispielhaften Systemstabilität ist KDE nicht mehr nur ein interessantes Experiment für Technologie-Enthusiasten, sondern auch zunehmend eine geeignete Wahl für den Einsatz in Firmenumgebungen.

KDE ist freie Software im Sinne der *Free Software Foundation* (FSF)[2]. Das bedeutet: Jeder hat das Recht, die Software für jeden Zweck einzusetzen, zu studieren, wie sie funktioniert, und an die eigenen Bedürfnisse anzupassen. Er darf sie verteilen, verbessern und die Verbesserungen der Öffentlichkeit zugänglich machen. Das bedeutet, dass KDE kostenlos verfügbar ist und dies auch immer bleiben wird. Der Begriff der Open-Source-Software ist identisch mit „freier Software".

„Environment" kann aber auch als die Projektumgebung selbst verstanden werden: als Netzwerk, in dem engagierte Personen gemeinsam arbeiten und ihre Kenntnisse einbringen. Mit mehr als 750 registrierten Entwicklern und vielen weiteren Mitwirkenden aus verschiedenen Ländern ist KDE eines der größten Open-Source-Projekte weltweit. Neben einigen hundert Softwareentwicklern bringen sich auch

2 *Free Software Foundation* – http://www.fsf-europe.de.

Übersetzer, Dokumentations-Autoren und Künstler aus vielen verschiedenen Ländern ein. Ihr persönlicher Einsatz für die Entwicklung freier Software ist einer der zentralen Aspekte, der diese heterogene Gruppe vereinigt.

Dieser Beitrag beschäftigt sich im Folgenden mit der Frage, was diese *community* ausmacht, was sie zusammenhält, sie sogar stetig anwachsen lässt und wie sie funktioniert.

2. Die Anfänge

Das KDE-Projekt wurde 1996 von Matthias Ettrich gegründet. Linux war damals gerade auf einem stabilen Stand angekommen und begann seinen Siegeszug in die Serverlandschaft. Leistungsfähige Tools und die außerordentliche Stabilität machten Linux schon 1996 zu einem wertvollen Betriebssystem. Allerdings war es noch völlig ungeeignet für „normale" Heim- und Office-Anwender ohne technische Kenntnisse.

Zwar waren bereits einige Windowmanager für den traditionellen X11-Desktop vorhanden. Doch beinhalteten diese selten mehr als eine Menüleiste mit Buttons für zur Verfügung stehende Programme – meist ein Browser, ein E-Mail-Programm, ein Editor und ein Kommandozeilen-Terminal. Elemente wie ein gemeinsames Drag&Drop-Protokoll, eine Zwischenablage für Copy&Paste zwischen verschiedenen Applikationen, gemeinsame Dialoge, einheitliche desktopweite Konfigurationen und ein gemeinsames Hilfesystem fehlten vollständig. Ebenso mangelte es an anderen wichtigen Elementen für die tägliche Arbeit: Netzwerktransparenz für einfachen Zugriff auf Netzlaufwerke auf Anwendungsebene, einfaches Drucken aus allen Anwendungen oder ein Komponenten-Framework. Für die Erstellung von GUI-(Graphical User Interface)-Applikationen wurden schwierig zu programmierende Toolkits wie Motif, Tcl/Tk oder sogar die X-API eingesetzt. Dies führte dazu, dass nur wenige grafische Anwendungen unter Linux verfügbar waren. Ein typischer PC-Desktop wurde mit Microsoft Windows betrieben.

Diesen Mangel sollte KDE beheben. Von Anfang an war das Ziel der Gründer, eben diesen nicht gerade geringen Anspruch zu erfüllen. Der neue Desktop sollte Linux nicht nur für IT-Spezialisten, sondern für jeden PC-Anwender benutzbar machen, genauso wie Microsoft Windows oder MacOS. Außerdem sollte er es ermöglichen, auf qualitativ hochwertigem Niveau mit modernen Methoden des Software-Engineerings Software entwickeln zu können.

Das erste Lebenszeichen von KDE datiert zurück auf den 14. Oktober 1996 und war in einigen Usenet-Gruppen zu lesen[3]. Matthias Ettrich, damals noch Informatik-Student an der Uni Tübingen, verschickte einen Text mit dem Aufruf, sich dem neuen Projekt anzuschließen. Das Dokument trug den Titel „New Project: Kool Desktop Environment (KDE). Programmers wanted". Dies beantwortet die häufig gestellte Frage, wofür das „K" in KDE steht. Heute wird das K nicht mehr als Abkürzung verwendet, sondern (vorläufig) nur noch allein stehend. Matthias Ettrich erklärt das Ziel folgendermaßen:

[3] KDE Gründungstext, 14. Okt. 1996 – u.a. http://www.hakubi.us/kdeannounce.html.

> [...] a GUI should offer a complete, graphical environment. It should allow a user to do his everyday tasks with it, like starting applications, reading mail, configuring his desktop, editing some files, delete some files, look at some pictures, etc. [...] A nice button with a nice „Editor"-icon isn't not at all a graphical user environment if it invokes „xterm -e vi".

KDE soll also sowohl mehr als ein Windowmanager sein, aber auch nicht nur eine Ansammlung von Anwendungen. An dem Projekt war seine Freundin nicht ganz unschuldig, wie folgende Textpassage belegt:

> [...] I really believed that is even yet possible with Linux until I configured my girlfriends box. [...] So one of the major goals is to provide a modern and common look&feel for all the applications. And this is exactly the reason, why this project is different from elder attempts.

Auch technisch stellt Matthias Ettrich direkt die Weichen. Um einen qualitativ hochwertigen Desktop zu erstellen, benötigt man eine qualitativ hochwertige Grundlage. Als objektorientierte C++-Bibliothek ermöglicht Qt die modulare Entwicklung nach modernen Methoden wie Entwurfsmustern:

> [...] I really recommand looking at this library [Qt]. It has [...] the power to become the leading library for free software development. And it's a way to escape the TCL/TK monsters that try to slow down all our processors and eat up our memory [...]

Er fügt dem Aufruf folgenden doch sehr optimistischen Zeitplan bei. Immerhin sollte es noch bis Mitte 1998 dauern, bis die erste stabile KDE-Version 1.0 verfügbar war.

> So there is a lot of work (and fun) to do! If you are interested, please join the mailing list. If we get about 20–30 people we could start. And probably before 24th December [1996] the net-community will give itself another nice and longtime-needed gift.

Kurz nach diesem ersten Aufruf wurde eine Mailingliste auf einem Rechner der Uni Tübingen eingerichtet, und es starteten sofort die Diskussionen über das neue Projekt. Nach vier Tagen stand bereits die erste Version für eine Startleiste zur Verfügung, es entstanden schnell die ersten Bibliotheken und ein Windowmanager. Torben Weis nahm sich der „file-manager-web-browser-desktop-manager-everything-else-as-well"-Anwendung an. Zunächst hieß diese Anwendung etwas langweilig „kfm" für „K File Manager" und wurde später in „Konqueror" umbenannt. Konqueror kann heute zwar nicht alles, aber als Dateimanager und Webbrowser ist er eines der vielseitigsten und flexibelsten Werkzeuge in KDE.

Zeitraum	*Ereignis*
14. Okt 1996	Gründung KDE-Projekt
Feb. 1997	Gründung der KDE-FreeQt-Foundation
Mai 1997	Linux-Kongress Würzburg: erster Vortrag über KDE
Okt. 1997	„c't"-Artikel über KDE von Matthias Kalle Dalheimer

Zeitraum	*Ereignis*
28. Aug.–1. Sep. 1997	KDE-One-Treffen in Arnsberg
Nov. 1997	Gründung des KDE e.V.
12. Juli 1998	KDE 1.0
7.–10. Okt. 1999	KDE Two, Erlangen
10.–20. Juli 2000	KDE Meeting Three Beta, Trysil
Sep. 2000	KDE 2.0
2001/2002	intensive Teilnahme an Messen: Linuxtag, Systems, CeBIT, LWCE
Feb.–März 2002	KDE Three, Nürnberg
25. Feb.–4. März, Sep. 2002	KDE Meeting Three, Erlangen
April 2002	KDE 3.0
Jan. 2003	KDE 3.1
Aug. 2003	Nove Hrady, Tschechien
Feb. 2004	KDE 3.2
geplant für 2004	KDE 3.3/4.0

Tabelle 1: Zeitplan

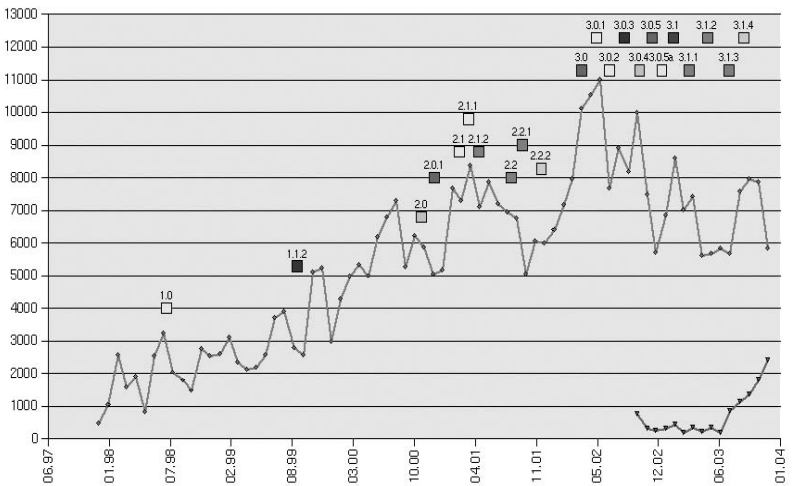


Abbildung 2: KDE CVS-(Concurrent Versioning System)-Statistik[4].

Das Diagramm zeigt die Anzahl der Codeänderungen an KDE pro Monat, die grau hinterlegten Boxen markieren die KDE-Releases. Im Oktober 2002 wurden „stille" Codeänderungen eingeführt, dies sind u.a. Übersetzungen. Diese sind in der unteren Linie dargestellt. Außerdem wurde die Auswertung so modifiziert, dass

4 KDE CVS-Statistik – http://kde.mcamen.de/statistics.html.

gleichzeitige Änderungen in verschiedenen Datei-Verzeichnissen nicht mehr getrennt gezählt werden. Beides legt nahe, dass die Steigerungsrate der Arbeit an KDE auch 10/02 weiter zugenommen hat.

KDE ist inzwischen ein sehr großes, wenn nicht sogar das größte *community*-Projekt in der Open-Source-Welt überhaupt. Es ist schwierig, die Größe von KDE genau zu benennen, aber die KDE-Programme bestehen inzwischen aus ungefähr drei Millionen Zeilen Code. Zum Vergleich – der Linux-Kernel 2.5.29 weist 3,1 Millionen Zeilen Code auf. Monatlich wurden zuletzt ca. 6000–8000 Änderungen vorgenommen, siehe dazu Abbildung 2. Das Diagramm zeigt die KDE-Releases im Laufe der Zeit. Heute wird KDE hauptsächlich auf GNU/Linux benutzt, aber auch auf Solaris, FreeBSD, OpenBSD, MacOS X und vielen mehr.

Noch schwieriger ist es, die Größe der KDE-*community* zu beziffern. Offiziell registriert sind derzeit knapp 900 Entwickler, von denen 300 im letzten Monat aktiv waren, und 450, die in den letzten 6 Monaten aktiv waren. Dazu kommen noch ungefähr 500 Übersetzer und Autoren der Dokumentation sowie Personen, die organisatorische Aufgaben übernehmen. Da hier keine Registrierung erfolgt, ist eine Zahlenangabe äußert schwierig. Unzählbar ist auch die Gruppe der Tester und der Personen, die für KDE Werbung machen, ohne im Projekt sichtbar zu werden. Der Übergang zum unbeteiligten Anwender ist oft fließend.

Ein Sättigungsgrad in der Größe der *community* ist nicht zu beobachten. Stattdessen sieht man eher eine Aufspaltung in Teilprojekte, sodass die Umgebung, in der man arbeitet, oft nicht mehr als vier bis 20 Personen umfasst. Typischerweise haben die Teilprojekte von KDE einen Projektleiter (Maintainer), der für einen individuellen Teil der Applikationen oder des Systems zuständig ist und die Arbeit koordiniert. Oftmals ist der Maintainer derjenige, der die erste Zeile Code für diesen Teilbereich geschrieben hat. Dabei wird ein Maintainer in den wenigsten Fällen formal eingesetzt.

3. KDE und der Rest der Welt

Wollen Entwickler ein Open-Source-Projekt nicht nur zum Selbstzweck betreiben, so sind sie auf zwei Dinge angewiesen. Dies sind Anwender, welche die Software zu schätzen wissen und Feedback geben können, und neue Entwickler. Da jeder Anwender auch ein potenzieller Kontributor ist, überschneiden sich die Zielgruppen zwangsläufig.

KDE hat den Schritt nach außen schon sehr früh gewagt. Der erste öffentliche Vortrag über KDE wurde von Matthias Ettrich auf dem Linux-Kongress in Würzburg im Mai 1997 mit überragendem Erfolg gehalten. Zwei Mitglieder eines konkurrierenden Projektes namens OffiX zogen daraufhin sogar ihren Vortrag zurück, und man hörte nie wieder von dem Projekt. Kurze Zeit später schrieb Matthias Kalle Dalheimer einen Artikel für das deutsche Computermagazin „c't". Noch heute berichten viele der KDE-Kernentwickler, dass sie auf Grund dieses Artikels zu KDE gestoßen sind.

In den nachfolgenden Jahren war die Außendarstellung von KDE hauptsächlich davon geprägt, dass auf Messen wie dem Linuxtag Vorträge gehalten und Artikel in

Computerzeitschriften geschrieben wurden. Auf einem der ersten Linuxtage – damals noch eine Universitätsveranstaltung von Studenten der Uni Kaiserslautern –, den ich 1999 selbst besucht habe, versuchte ich, in einen Vortrag von Torben Weis über KOffice zu gelangen. Der größte Vortragssaal war so überfüllt, dass ich nur gelegentlich über die Köpfe der anderen Besucher hinweg einen Blick auf den Vortragenden erhaschen konnte. Dieser Andrang und das zunehmende öffentliche Interesse bewirkten, dass die Messetätigkeit mit zunehmend stärkerer Präsenz ausgeweitet wurde.

Da die Außenwirkung des Projektes wichtig ist, unterhält das KDE-Projekt eine Reihe von Web-Diensten, die zum Teil der Selbstdarstellung dienen, zum Teil als Nachrichtendienste arbeiten, um Interessierten einen Einblick in alle Aktivitäten rund um das KDE-Projekt zu bieten. Herausragend ist hier „The Dot" (dot.kde.org). Dieses stark durch Slashdot inspirierte Nachrichtensystem trägt in professioneller Art und Weise Nachrichten aus allen Bereichen des Projektes zusammen, bringt Interviews mit prominenten Entwicklern und Persönlichkeiten der Open-Source- und Free-Software-Szene und stellt gleichzeitig eine Diskussionsplattform für Nutzer und Entwickler dar. Wie alles in KDE ist auch der „Dot" ein reines *community*-Projekt. Gesponsert werden durch Dritte lediglich Bandbreite und der Server, alle Arbeit wird von Freiwilligen übernommen.

Abseits der Öffentlichkeit werden außerdem freundschaftliche Beziehungen zu den verschiedensten Projekten gepflegt. Eine Messe dient gerne als Anlass, um nach getaner Arbeit beim gemeinsamen Abendessen und einigen Bieren zusammen zu klönen, Neuigkeiten aus der sonstigen Linux-*community* zu erfahren und Ideen zu spinnen. Während des Besucherandrangs tagsüber bleibt oft nur wenig Zeit, um mal zu den anderen Ständen zu gehen. Auch die Social Events der Messeveranstalter sind beliebte Treffpunkte, zumal das gute Essen doch etwas für die viele Arbeit am Stand entschädigt.

Die Beziehungen beschränken sich aber nicht nur auf die soziale Ebene. KDE ist beispielsweise ein vorantreibendes Projektmitglied der Freedesktop-Gruppe[5], in der auch der Konkurrenzdesktop zu KDE namens GNOME Mitglied ist. Freedesktop.org hat sich zum Ziel gesetzt, Kerntechnologien auf dem Desktop zu standardisieren und zu vereinheitlichen, damit die Interoperabilität zwischen verschiedenen Desktop-Systemen und GUI-Bibliotheken zugunsten des Anwenders verbessert wird. Auch bei dem Thema des barrierefreien Desktops wird gemeinsame Technologie eingesetzt. Letztendlich versteht sich KDE aber heute als integrativer Desktop, der alle Anstrengungen auf dem Desktop zusammenführen will.

Dies war nicht immer selbstverständlich. GNOME wurde 1997 ursprünglich als gegnerisches Konkurrenzprojekt zu KDE gegründet. Einer der größten Kritikpunkte gegenüber KDE war, dass die Lizenz des verwendeten Qt-Toolkits nicht mit einer von der FSF anerkannten Definition für freie Software vereinbar war. KDE selbst stand immer unter der GPL (*General Public License*), die KDE Libraries unter der LGPL (*Lesser General Public License*). Die GPL-Lizenz ist die offziell anerkannte Lizenz der FSF, welche die oben beschriebenen Freiheiten garantiert und diese da-

[5] Vgl. http://www.freedesktop.org.

vor schützt, dass sie verloren gehen. Die Firma Trolltech, die Qt entwickelt, stellte ebenfalls den Quellcode kostenlos zur Verfügung, was die KDE-Gründer mit ihrem meist sehr pragmatischen Ansatz auch für Projekte nutzten. Zugleich erlegte aber Trolltech seinen Nutzern einige Einschränkungen auf, zum Beispiel durfte Qt nicht verändert werden, und es durften keine veränderten Versionen in Umlauf gebracht werden.

Mit dem raschen weltweiten Erfolg von KDE wurde diese „Schwäche" in seinen Qt-Grundlagen immer deutlicher. Viele Kräfte in der wachsenden Free-Software-Bewegung sahen die Gefahr, dass das kostenlose Qt eines Tages von Trolltech oder einem böswilligen Monopolisten, der Trolltech einfach aufkauft, vom Markt genommen werden könnte. KDE würde auf dem Trockenen sitzen, und Traum wie Wirklichkeit vom schönen neuen Unix-Desktop wären dahin.

Einige Programmierer riefen deshalb das GNOME-Projekt ins Leben, das „GNU Network Object Model Environment". GNOME sollte technisch dieselben Ziele verfolgen wie KDE, jedoch unter einer von der FSF anerkannten freien Lizenz stehen. Die Debatte zwischen GNOME- und KDE-Leuten war anfangs nicht gerade freundschaftlich, sondern sogar extrem polarisiert, und es wurde mit großer Erbitterung „gekämpft".

Gleichzeitig hatte sich die KDE-Gemeinde selber um eine Lösung des Problems bemüht. Bereits im Februar 1997 wurde die KDE-FreeQt-Foundation[6] gegründet, welche mit Trolltech einen Vertrag schloss. Dieser sieht vor, dass die Freiheit von Qt gewissermaßen auf Ewigkeiten garantiert ist. Sie ist auch davor geschützt, dass ein mächtiger Konkurrent Trolltech aufkauft, denn in diesem Falle „erbt" das KDE-Projekt den Sourcecode und muss ihn unter einer freien Lizenz freigeben. Auch eine dauerhafte Innovation ist gesichert. Sollte Trolltech einmal zwölf Monate lang keine neue Version von Qt mehr herausgeben, kann KDE die Sourcen ebenfalls frei veröffentlichen.

In der Version 2.0 wurde Qt zudem von Trolltech zur freien Softwareentwicklung unter die GPL gestellt. Zum Zwecke proprietärer, kommerzieller Entwicklung kann man jederzeit Lizenzen kaufen.

Die freie Software-Welt hat inzwischen längst wieder ihren Frieden mit KDE gemacht. Das Verhältnis zu GNOME hat sich gründlich gewandelt: Der Wettbewerb ist heute eher von „Sportsgeist" geprägt anstelle alter Feindseligkeiten:

Man arbeitet an vielen Stellen zusammen, legt gemeinsame Standards fest, man kooperiert bei der Erstellung beiderseits genutzter Softwarebibliotheken, und man hat sich zu dem oben erwähnten Projekt Freedesktop.org zusammengeschlossen.

KDE hat nicht nur eine Stärkung seiner Lizenzbasis erhalten, die indirekt durch die GNOME-Entwicklung begünstigt wurde. Auch die Open-Source-Bewegung wird durch Konkurrenz belebt, und weder KDE noch GNOME wären ohne den gegenseitigen Ansporn heute da, wo sie sind. Allerdings hat KDE auf Grund seiner inneren Technologie, die es u.a. dem ausgereiften Qt-Toolkit der modernen Entwicklungsmethoden verdankt, immer noch einen immensen Vorsprung vor GNOME.

6 Vgl. http://www.kde.org/whatiskde/kdefreeqtfoundation.php.

Eine besondere Anerkennung war es deshalb auch, als Apple verkündete, die zentralen KDE-Bibliotheken für die Darstellung von Webseiten und für Javascript verwenden zu wollen, um einen eigenen Browser für sein MacOS-X-Betriebssystem zu bauen. Diese Bibliotheken sind auch die Herzstücke des KDE-Browsers Konqueror. Apple sah in der schnellen und genauen Webrendering-Engine und in der Standardtreue, die von den KDE-Programmen eingehalten wird, die entscheidenden Vorteile. Ganz im Sinne der LGPL-Lizenz liefert Apple jetzt auch seit über einem Jahr die an den KDE-Bibliotheken vorgenommenen eigenen Verbesserungen an KDE zurück.

4. Leben in der community

Von Anfang an fand die Entwicklung von KDE im Internet statt. Begonnen hat alles mit einer einzelnen Mailingliste und einem Sourcecode-Repository auf einem Rechner an der Universität Tübingen. Ein Sourcecode-Repository (CVS) ermöglicht den parallelen Zugriff auf den Sourcecode der Programme über das Internet. So ist es möglich, dass Entwickler aus den verschiedensten Teilen der Welt gleichzeitig an dem gleichen Projekt zusammenarbeiten. Auf Grund der internationalen Mitgliedschaft ist die Projektsprache durchgängig Englisch. Mit dem Wachstum des Projektes kristallisierten sich schnell Teilprojekte wie Window- und Filemanager heraus, und es wurde notwendig, die Kommunikationsmittel entsprechend anzupassen. Die Kapazität des zentralen Servers, einer ausrangierten Arbeitsstation, reichte schon bald nicht mehr aus.

Inzwischen steht für die Infrastrukur des Projekts eine Reihe von Servern mit Mailinglisten, Web- und Download-Angeboten mit ausreichender Verfügbarkeit und Bandbreite bereit. Ein Teil der Server steht an deutschen Hochschulen, die das Projekt von Anfang an durch die kostenlose Ressource Internet unterstützt und es dadurch erst möglich gemacht haben. Ohne Internet gäbe es kein KDE. Zusätzlich zu insgesamt neun zentralen KDE-eigenen Servern gibt es weltweit weitere Server, die Inhalte der KDE-Server vervielfältigen, um so die Belastung vor allem kurz nach der Freigabe einer neuen KDE-Version auf viele Rechner zu verteilen. Weltweit gibt es ca. 25 alternative Webserver und über 70 Server zum Download der Programme in über 30 Ländern.

Das KDE-Projekt besteht aus 22 aktiven Teilprojekten und über 80 Mailinglisten, die zumeist wiederum einzelnen Projekten zugeordnet sind. Größere Projekte besitzen dabei durchaus mehrere Mailinglisten, die sich speziell mit besonderen Aspekten wie Anwendersupport oder Entwicklungsfragen beschäftigen. Dadurch ist das Nachrichtenaufkommen gut kanalisiert und die Qualität der technischen Beiträge sehr hoch.

Eine besondere Rolle nehmen die Mailinglisten „kde-devel“ und „kde-core-devel“ ein. Sie gehören keinem speziellen Teilprojekt an, sondern dienen der projektübergreifenden Kommunikation. Die Liste „kde-devel@kde.org“ steht allen Entwicklern offen, sie ist eine Art KDE-internes technisches Supportforum. Hier kommen auch besonders Entwickler zu Wort, die an Projekten außerhalb der KDE-

eigenen Strukturen arbeiten. Zumeist melden sich hier Applikations-Entwickler, die auf ein Problem gestoßen und auf der Suche nach einer eleganten Lösung sind.

Die Themenauswahl auf „kde-core-devel@kde.org“ ist dagegen strikt auf technische Diskussionen beschränkt, die sich auf die Entwicklung der KDE-Plattform (Bibliotheken und Kernkomponenten) beziehen. Die Mailingliste ist zwar öffentlich und jeder kann sie abonnieren, nur eine ausgewählte Gruppe von Entwicklern kann jedoch Nachrichten schreiben. Es ist nicht ganz einfach, in diese erlesene Gruppe aufgenommen zu werden. Nur Entwickler, die maßgeblich an der Weiterentwicklung der KDE-Plattform mitarbeiten und sich dabei eine gewisse Seniorität erarbeitet haben, erhalten das Schreibrecht. Die Qualität der Beiträge ist daher in aller Regel sehr hoch. „Flamewars“, in denen sich Teilnehmer gegenseitig beschimpfen, sind selten und werden meist schnell unterbunden. Trotzdem ist der Ton auf allen Mailinglisten stets locker und informell. Letztlich gilt die Regel, dass bei Konflikten derjenige entscheidet, der die Umsetzung durchführt.

Doch Mailinglisten sind bei weitem nicht das einzige Kommunikationsmittel. Manchmal ist eine „synchronere“ Kommunikation notwendig, und so werden auch Instant-Messenger-Dienste wie ICQ („I Seek You“) und der Internet Relay Chat (IRC) intensiv genutzt. Interessant dabei ist, dass die Intensität der Kommunikation sich meist direkt auf die Effizienz der Entwickler auswirkt. Offensichtlich ist gegenseitiger Ansporn eine der stärksten Triebkräfte innerhalb des Projektes. Unter diesem Gesichtspunkt erlangen die regelmäßigen Entwicklertreffen eine besondere, herausragende Bedeutung.

Offensichtlich benötigt ein Projekt dieser Größe irgendeine Form von Koordination. Hier spielt der KDE e.V. als deutscher Verein mit beantragter Gemeinnützigkeit eine aktive Rolle. Der Verein organisiert die Kernentwickler unter seinem Dach und kümmert sich um finanzielle und rechtliche Belange der *community*. Dabei geht es nicht um die technische Leitung des Projektes. Eines der Erfolgsrezepte des KDE-Projektes liegt darin, die Kreativität der Entwickler nicht unnötig durch Formalismen einzuschränken. Daher konzentriert sich der KDE e.V. darauf, die Arbeit der Entwickler durch die Bereitstellung von Infrastruktur zu unterstützen. Gleichzeitig dient er als zentrale Anlaufstelle für Sponsoren. Die Spenden, die an den Verein fließen, werden dafür eingesetzt, Reisekosten von KDE-Entwicklern zu Messen und Treffen mit anderen Entwicklern zu decken. Außerdem führte der KDE e.V. 2003 zum ersten Mal in Nove Hrady, Tschechien, eine KDE-Konferenz der Kontributoren durch.

Gegründet wurde der KDE e.V. im November 1997. Anlass war die Durchführung des ersten KDE-Treffens in Arnsberg im August des gleichen Jahres. Matthias Kalle Dalheimer, damals noch Student, hatte die Finanzierung des Treffens mit einem Budget von 14000 DM über sein privates Konto abgewickelt und weigerte sich, dies noch einmal zu tun. Die Gründung des Vereins sollte einen rechtlichen Rahmen für weitere Aktivitäten schaffen und helfen, die finanziellen Risiken für die Organisatoren abzufedern.

Regelmäßigen Entwicklertreffen kommt im KDE-Projekt eine große Bedeutung zu. Erfahrungen aus der Vergangenheit haben immer wieder gezeigt, dass nichts die Tätigkeit der Projektangehörigen so sehr anspornt wie die unmittelbare Zusammen-

arbeit mit anderen Entwicklern. Dabei ist es gar nicht notwendig, formelle Treffen abzuhalten; die Zusammenarbeit ist auch über das Internet sehr eng, sie gewinnt lediglich eine besondere Qualität, wenn man sich auf Rufweite im gleichen Raum befindet. Schon oft haben sich solche Treffen unmittelbar vor einem Release als extrem produktiv herausgestellt. Sie sind die intensivsten Arbeitsphasen begleitet von kreativen und innovativen Problemlösungen und Neuentwicklungen. Diese Produktivität kann durchaus ein zweischneidiges Schwert sein. Die Veröffentlichung des Release 3.2 musste nach dem Treffen in Nove Hrady um einen Monat verschoben werden, bis der Sturm der „CVS-Commits" (Hinterlegen von neuem Sourcecode oder Änderungen auf dem zentralen Server) auf ein überschaubares Maß abgeflaut war.

Entwicklertreffen haben eine gewisse Eigendynamik. Sie enstehen zum Teil spontan auf Messen, sobald eine ausreichend große Ansammlung von KDElern einen Kristallisationskeim gebildet hat. Zu Beginn litten die Projektpräsentationen am KDE-Messestand häufig darunter, dass Entwickler an den Demorechnern sich mehr mit ihren Programmen beschäftigten als mit den Besuchern. Seit der KDE e.V. die meisten der KDE-Messeauftritte in Deutschland organisiert, wurde daher stets darauf geachtet, auf dem eigenen Stand eine ausreichend große Fläche für spontane Meetings zufällig eintreffender Entwickler zu reservieren. Beliebt sind diese „Hacking-Areas" natürlich auch bei den Messebesuchern. So bilden sich des Öfteren große Trauben um einen Bildschirm, an dem gerade ein KDE-Entwickler etwas Interessantes ausprobiert. Nicht selten werden auf Zuruf neue Features implementiert oder Fehler behoben.

Das Bedürfnis, sich auszutauschen, weitet sich auch auf die Abende während einer Messe aus. Manch einer wäre nach einer Messe dem Verhungern nahe, würden die Organisatoren sich nicht um gemeinsames Essen kümmern. Auf dem Linuxtag 2003 besuchte das KDE-Team zusammen mit Mitgliedern einiger befreundeter Projekte, u.a. GNOME und Debian, ein Lokal, in dem vorher ein Buffet bestellt worden war. Immerhin wurde nach dem Essen eine Anstandszeit von 30 Minuten eingehalten, bis die ersten Laptops zu sehen waren.

Abbildung 3: Der KDE-Stand auf dem Linuxtag 2002 vereinigte Programmierer in einer „Hacker-Area" (im vorderen Teil zu sehen) für das spontane Zusammentreffen der Entwickler und Demostände, an denen Besuchern KDE vorgeführt wurde.

Dass diese Kombination aus Messe und „Hacking-Event" sich bewährt, zeigen die Teilnehmerzahlen aus dem Projekt, die bei 40–60 Entwicklern liegen. Die Möglichkeit, auf dem Linuxtag auch Entwickler anderer Projekte persönlich zu treffen und sich mit ihnen auszutauschen, trägt sicherlich auch dazu bei. Bei der Gelegenheit wird auch kritisch beäugt, was die Konkurrenz zu bieten hat. Trotzdem ist festzuhalten, dass KDE trotz allem eine freundschaftliche Beziehung zu allen Open-Source-Projekten und auch zu GNOME pflegt.

Dass solche Kurzmeetings und auch allgemeine Entwickler-Treffen während Messen nicht ausreichen, wurde bereits auf der ersten Messepräsenz 1997 auf dem Linux-Kongress in Würzburg festgestellt. Deshalb wurde beschlossen, noch im selben Jahr ein reines KDE-Treffen durchzuführen. Es trafen sich denn auch im August 14 KDE-Entwickler in Arnsberg, und in den drei Tagen wurden einige Meilensteine zurückgelegt, u. a. die Mimetype-Erkennung, Internationalisierungs-Unterstützung für die Übersetzung in andere Sprachen, ein stabiles Drag&Drop und die Defintion eines Filesystem-Standards für die KDE-Installation. Auch das zweite KDE-Treffen 1998 in Erlangen war ähnlich produktiv und wegweisend für KDE. In Vorbereitung auf den nächsten großen Umbruch in KDE in Form von Version 2.0 begann man, die Desktop-Infrastruktur, die sich in KDE 1 als problematisch erwiesen hatte, zu überarbeiten. Das schwierig zu programmierende, fehleranfällige und schwergewichtige „CORBA" (Common Object Request Broker Architecture), welches zur Kommunikation zwischen Programmen und zur modularen Programmie-

rung diente, wurde dabei entfernt und durch eine Eigenentwicklung ersetzt: „KParts"-Komponenten sind wesentlich flexibler, schlanker, schneller und von allen Entwicklern leichter zu verstehen und zu verwenden. Matthias Ettrich und Preston Brown behaupteten nach einigen Bierchen, sie könnten in einer Nacht ein neues Kommunikationsprotokoll als Ersatz für CORBA schreiben – am nächsten Tag mussten sie beweisen, dass sie es ernst gemeint hatten. Und so entstand eine einfache und schnelle Skriptinganbindung für KDE-Anwendungen namens „DCOP", das Desktop COmmunication Protocol ...

Die Treffen KDE Three Beta und KDE Three standen unter dem Motto „Bugfixing". Bugfixing bedeutet Fehlerbehebung – dies ist die jeweilige Hauptaktivität in der letzten Phase, bevor eine KDE-Version veröffentlicht und freigegeben wird. Die KDE-Releases laufen nach einem festen Schema ab, währenddessen immer eine Serie von Testversionen veröffentlicht wird. Je nach Stabilität haben sie den Beinamen „Alpha", „Beta" oder „Release Candidate".

Vor Beginn eines Release-Ablaufs wird eine Liste mit „Features" erstellt – neue Funktionalitäten, die in die nächste Version eingehen sollen und erst programmiert werden müssen. Ab einem vorgegebenen Zeitpunkt dürfen keine neuen Features mehr in KDE eingebracht werden. Noch nicht begonnene Features werden in die nächste KDE-Version verschoben. Die nun folgende Phase wird „Feature-Freeze" genannt. Unvollständig implementierte Features werden fertig programmiert. Danach folgt der „String-Freeze". Hier dürfen bis zum Release keine für den Anwender sichtbaren Texte geändert werden. Dies ermöglicht den Übersetzer-Teams, ihre Arbeit zu tun. KDE besteht immerhin aus ca. 120000 Textelementen. KDE 3.1 ist derzeit in 42 Sprachen komplett übersetzt (darunter viele, die von Microsoft nicht berücksichtigt werden!) und weitere 35 bereits teilweise. Für jede dieser Sprachen gibt es kleine oder größere muttersprachliche Übersetzungs-Teams, die selbstständig arbeiten und von einem Koordinator geleitet werden. Auch die Dokumentations-Autoren kommen nun zum Zuge und erstellen Screenshots als Ergänzung zum Text. Die letzte Phase besteht nur noch aus „Bugfixing". Hier werden nur noch Fehler (bugs) behoben, die Anwender in eine eigens dafür zur Verfügung stehenden Datenbank unter http://bugs.kde.org/ eingetragen haben. Diese Fehlerdatenbank steht jederzeit offen und kann befüllt (oder auch nur gelesen) werden. Bugfixing ist die vielleicht „langweiligste" Arbeit für Entwickler und doch mit die wichtigste, damit KDE für den Anwender letztlich benutzbar ist.

Dieses Verfahren für einen KDE-Release mag einem recht formal vorkommen. Es wurde über die verschiedenen KDE-Versionen von den Entwicklern im Konsens entwickelt. Bewährt hat sich außerdem, dass für jede größere Version von KDE ein so genannter „Release Dude" bestimmt wird. Dies ist der „Dumme", der für die Koordination des Releases und die Einhaltung des Zeitplans verantwortlich ist. Zur Belohnung bekommt er einen Ehrenplatz in der KDE-Gemeinde.

Das letzte KDE-Treffen fand im August 2003 in Nove Hrady in Tschechien statt. Eingeleitet wurde das Treffen von einer Mitgliederversammlung des KDE e.V. und – als öffentlichem Teil – einer zweitägigen Konferenz. Auch hier fand ein emsiger Austausch statt. Bei manchem Teilnehmer begann dieser sogar schon auf der Hinfahrt: Eine Gruppe von Entwicklern kam in einem gemieteten

Reisebus, der auf einer Route über Frankfurt, Darmstadt und Nürnberg Leute einsammelte. Der Bus war sofort der erste „Hacking-Workshop", die ersten Features wurden implementiert, bis ... die Laptop-Batterien leider viel zu schnell leer waren. Besonderes Aufsehen beim nicht-technischen Konferenzteam vor Ort erregten die ersten Entwickler, die in Nove Hrady eintrafen und – statt sich für Essen und Unterkunft zu interessieren – einen großen Computer hereintrugen und nach einer Steckdose fragten.

Abbildung 4: Teilnehmer der KDE-Konferenz 2003 in Nove Hrady

5. Wer steckt hinter KDE?

Die Gruppe der aktiven KDE-Mitglieder ist sehr heterogen und besteht sowohl aus Studenten als auch Berufstätigen, vornehmlich aus den Bereichen Informatik, Natur- und Ingenieurwissenschaften, aber auch aus vielen anderen Fächern.

Laut einer soziologischen Studie, durchgeführt von Dipl.-Soz. Andreas Brandt (Brandt 2003), weist ein typischer KDEler folgende Eigenschaften auf. Er ist zwischen 20 und 30 Jahre alt, männlich (das ist wohl typisch für den gesamten Bereich der Informatik, ganz besonders aber leider für die Open-Source-Communities), studiert, aus Deutschland, Single und arbeitet ca. 15 Stunden pro Woche während seiner Freizeit an KDE.

Für ihre KDE-Arbeit bezahlt werden nur sehr wenige. Von den vielen Hunderten Mitwirkenden sind es nur ungefähr zehn Projektmitglieder. Einige wenige haben inzwischen außerdem den Schritt in die Selbstständigkeit gewagt. Sie bieten auf dem wachsenden Markt rund um den Linuxdesktop Programmierleistungen oder Bera-

tung und Support für größere Firmeninstallationen an. Als treibende Kraft hinter dem Projekt ist jedoch keine Firma zu finden. Dies erklärt, warum die Arbeit des Projekts in Bezug auf Werbung und Marketing wohl eher wenig ausgeprägt ist. Ohne große Marketing-Budgets wird man für KDE nie große Werbekampagnen finden. Das KDE-Projekt setzt stattdessen auf die Arbeit der vielen Projektteilnehmer, die wie Ameisen im Kleinen Werbung machen, damit sich die Möglichkeiten, die dieser Desktop bietet, herumsprechen.

Als Hauptmotivation der meisten Projektteilnehmer kommt eine finanzielle Motivation nicht zum Tragen, sondern eher der Spaß an der Sache und vielleicht auch die Anerkennung. Auf der Konferenz in Nove Hrady trug ein Entwickler ein T-Shirt mit der Aufschrift „code poet" – dies trifft das Selbstverständnis vieler Entwickler wohl sehr gut. Es geht um zielorientiertes und praxisorientiertes Arbeiten mit dem Ziel, schöne Lösungen zu entwicklen. Das KDE-Manifesto[7] führt hierzu den Begriff „commitment to excellence" auf – Verpflichtung zu hervorragender Arbeit.

Die in diesem Dokument beschriebenen Regeln des Projekts – komplettiert von vielen ungeschriebenen – haben zum Ziel, eine fortlaufende hohe Systemqualität zu gewährleisten und gleichzeitig so demokratisch und offen wie möglich zu sein. Im Projekt kann zwar eine Kerngruppe von „wichtigen" Entwicklern identifiziert werden, aber kein „wohlwollender Dikator" (benevolent dictator), wie Eric S. Raymond ihn in seinem Buch (Raymond 2001) beschreibt. Vielmehr basiert die Gemeinschaft auf einem „Netz des Vertrauens" aus solchen Personen, die schon länger am Projekt beteiligt sind. Dieses Vertrauen entsteht hauptsächlich aus der Reputation und der Menge und Qualität der aktiven Beiträge zu dem Projekt. Also auch, wie loyal die Mitglieder zu dem Projekt sind.

Doch diese Angaben sind eher abstrakt und geben nur ein schwaches Bild der *community* wieder. Als Beispiel möchte ich daher berichten, auf welchen Wegen ich selber und ein weiteres Projektmitglied zu KDE gestoßen sind.

Das erste Mal hatte ich von KDE 1998 an der Universität gehört. Damals wurden an unserem Fachgebiet SUN-Workstations als normale Arbeitsplätze eingesetzt. Ein Administrator einer benachbarten Arbeitsgruppe hatte eine neue Desktop-Software namens KDE aufgetan. Auf einmal war Farbe im Spiel, Fenster mit Icons, eine Menüleiste, in die man Programme platzieren konnte, und komfortable Programme zum Browsen und E-Mail-Schreiben. Für Unix war das eine unglaubliche Bereicherung. Wenige Zeit später stand der Kauf eines Privatrechners inklusive Linux an. Auch hier begegnete mir wieder die komfortable Oberfläche namens KDE. Im Vergleich zu Windows 3.11 war das ein enormer Fortschritt, insbesondere durch die Kombination mit dem flexiblen Betriebssystem GNU/Linux.

Als begeisterte Anwenderin traf ich auf dem Linuxtag 1999 in Kaiserlautern zum ersten Mal auf „echte Entwickler". Ich hatte inzwischen mit einer Promotion begonnen und am Fachgebiet die Systemadministration übernommen. Bei dem Versuch, KDE auf eine aktuellere Version aufzurüsten – die Webseite des KDE-Projekts verhieß viele Neuerungen –, stellte sich heraus, dass Solaris keine einfache

7 KDE-Manifesto – http://www.kde.org/whatiskde/kdemanifesto.php.

Plattform ist und manches nicht so einfach funktioniert wie unter Linux. Also befragte ich einen Entwickler zu meinen Problemen, ohne dass er mir weiterhelfen konnte. Offensichtlich hatte ich doch sehr eindringlich gefragt, sodass sich mein Gesprächspartner auf dem nächsten Linuxtag 2000 wieder an mich erinnerte. Ich hatte inzwischen gelernt, wie Open Source funktioniert, wenn sich die Uni keine teuren Support-Verträge leisten kann (2000 gab es zudem nur wenig Anbieter dafür): Wenn einem keiner helfen kann, dann muss man sich eben selber helfen. Und da man mit KDE eine Unmenge von Software bekommt, ist es nur selbstverständlich, dass man das bisschen, das man selber machen kann, auch wieder der Allgemeinheit zur Verfügung stellt. Im Herbst 2000 wurde ich dann von Torsten Rahn und Ralf Nolden, zwei deutschen KDE-Entwicklern, gefragt, ob ich nicht auf der Messe „Systems 2000" am KDE-Stand helfen möchte. Damit begann meine „Karriere" im KDE-Projekt in einer Funktion, die in der Industrie „Event-Managerin" heißen würde.

Auf der Systems 2000 lernte ich einen anderen KDE-Anwender namens Kurt Pfeifle kennen, der für eine bekannte Firma arbeitet, die „große" Drucker im gewerblichen und industriellen Bereich vertreibt, wartet und in Netzwerke einbindet. Er sprach immer wieder über das Drucken unter KDE und Linux im Allgemeinen und dass dafür mehr getan werden müsste. Da er selber kein Programmierer ist, spendete er 2001 sogar einen Betrag an den KDE e. V., damit jemand eine entsprechende Funktionalität implementiert. Allerdings bezahlt der KDE e. V. keine Entwickler (die Ressourcen sind ohnehin viel zu klein), und bei einem Projekt auf freiwilliger Basis kann auch niemand zu solch einer Aufgabe verpflichtet werden. Die Spende wurde demnach für die üblichen Ausgaben des KDE, e. V. wie Reisekosten verwendet. Ungefähr zur gleichen Zeit stieß glücklicherweise Michael Goffioul zum KDE-Projekt. Michael war ein begeisterter belgischer KDE-Anwender, allerdings einer mit profunden Programmierkenntnissen. Er machte sich – ohne Kenntnis von der Spende – an die Implementierung eines komfortablen Drucksystems für KDE. Die beiden fanden sich schnell und sind bis heute noch ein einzigartiges Gespann aus begeistertem Entwickler und Tester, Ideengeber und Dokumentierer. Kurt Pfeifle ist einer der wichtigsten Promoter für KDE und beschäftigt sich längst mit mehr als nur dem Drucksystem.

6. Quo vadis?

So hat jeder seine persönliche Geschichte, wie er zu KDE gekommen und seine eigene Motivation, warum er noch dabei ist. Gemeinsam ist aber allen die Begeisterung für die Software und das Projekt und der Spaß, den besten Desktop der Welt „für sich selbst" zu erstellen oder einfach nur bei einer großartigen Entwicklergemeinde „dabei zu sein". Das Arbeitsfeld bei der Erstellung eines Desktops ist so vielfältig, dass jeder seinen Platz findet.

Und neue Ufer werden jetzt von vielen KDE-Mitwirkenden auch angestrebt: Sie kommen in einem Slogan zum Ausdruck, der das alte Kürzel „KDE" neu beschreibt – als „KDE Desktop for the Enterprise" oder „Korporate Desktop Environment". KDE spielt die zentrale Rolle, wenn Linux seinen Marsch aus den klima-

tisierten Serverräumen hinaus unternimmt – hinein in die Büros und auf die Workstations in Unternehmen, Behörden und Organisationen.

Literatur

Brandt, Andreas (2003): *Structure and Organisation of KDE,* KDE Contributors Conference 2003,
online http://events.kde.org/info/kastle/conference_program.phtml.
Raymond, Eric S. (2001): *The cathedral and the bazaar*, O'Reilly 2001.

Steven Weber: The Success of Open Source

HENDRIK SCHEIDER

In acht Kapiteln umkreist der Autor der voraussichtlich 2004 erscheinenden Publikation „The Success of Open Source“[1] das Softwareentwicklungs- und Eigentumsmodell Open Source. Mit einer fachübergreifenden Analyse, die nicht nur auf der Informatik fußt, sondern auch auf Soziologie, Politologie, Ökonomik und Philosophie, leistet Weber zweierlei. Sowohl erklärt er das empirische Phänomen, dass es ein loser Verbund von professionellen Programmierern und Hobbyisten geschafft hat, wertvolle und komplexe Software hervorzubringen in einer weltweit einmaligen kollektiven Anstrengung. Doch er verallgemeinert auch den Ansatz im Sinne einer echten Theorie, die mehr leisten muss, als das Erscheinungsbild zu umfassen, aus dem sie abgeleitet wurde.

So liegt auch die Vermutung nahe, dass Weber kein Informatiker ist. Und in der Tat sieht er die Informatik und ihre Begleiterscheinungen allein als Gegenstand der Betrachtung. Seinen ersten akademischen Grad erwarb Weber in Geschichte, nachdem er in Mathematik gescheitert war. Anschließend ging er auf die Medical School in Stanford, wo ihm die wissenschaftlich-theoretische Arbeit der ersten Jahre gefiel. Doch die praktisch orientierten höheren Semester ließen ihn sich erneut umorientieren, sodass er schließlich eher zufällig beim Institut für Politikwissenschaften landete. Ganz offensichtlich ein glücklicher Zufall, denn 1989 wurde ihm dort der Doktortitel verliehen.

Heute doziert Professor Weber an der Universität von Kalifornien, Berkeley, und sein besonderes wissenschaftliches Interesse gilt der internationalen politischen Ökonomie, dem politischen und sozialen Wandel in der „new economy“ und der europäischen Integration. Zu seinen Publikationen zählen unter anderem das von ihm herausgegebene Buch „Globalisierung und die europäische politische Ökonomie“ (erschienen bei Columbia University Press) sowie zahlreiche Artikel zur amerikanischen Außenpolitik, der politischen Ökonomie von Handel und Finanzen, der politischen Landschaft nach dem Kalten Krieg und zur Europäischen Integration. Außerdem war er wissenschaftlicher Berater der Europäischen Bank für Wiederaufbau und Entwicklung.

Mit GNU/Linux und damit der Entwicklungsphilosophie Open Source kam Weber zum ersten Mal 1999, also aus computerhistorischer Sicht recht spät, in Berührung. Der Bottom-up-Ansatz und das kollektive Leistungsvermögen bei scheinbar gänzlich fehlender institutionalisierter Kontrolle haben ihn sogleich fasziniert, sodass er begann, nicht nur Linux, sondern auch die vielen anderen wichtigen Projekte näher zu betrachten. Und bei der empirischen Aufarbeitung des Phänomens

---

[1] Steven Weber: The Succes of Open Source. Harvard University Press 2004, vorliegend als Buchmanuskript.

kam ihm als Historiker gelegen, dass die *community* ihre Kommunikation minutiös in Log-Dateien und E-Mails archiviert.

So schließt an ein einleitendes erstes Kapitel des Buches eine äußerst detailliert recherchierte Geschichte der Softwareentwicklung im Allgemeinen und Open Source im Besonderen an. Dabei reiht er keineswegs Ereignis an Ereignis, sondern setzt sie in Relation zueinander, arbeitet Konsequenzen heraus, ohne sie schon gleich zu interpretieren. Und doch bereitet er jede Information darauf vor, sie in seine Erklärung und Theorie einzubauen, die ab Kapitel fünf folgen werden. Das dritte Kapitel widmet Weber einer Faktensammlung über die Besonderheiten von Open-Source-Software. Er stellt die selbst verfassten ethischen Grundprinzipien der Free Software Foundation und der OSS-Gemeinde vor, präsentiert einige Grafiken über Herkunft und Hintergrund der Entwickler und kontrastiert die Idealtypen sowohl der hierarchischen wie der OSS-Entwicklungskultur. Dass es Free Software schaffen würde, zu einer weltweit beachteten Bewegung zu werden, war Anfang der 90er Jahre gar nicht gewiss. Zu eingespielt war das klassische Modell und drohte den alternativen Ansatz gänzlich zu marginalisieren. Diese sich zuspitzende Situation während der letzten fünfzehn Jahre dokumentiert Kapitel vier, das mit dem Hinweis auf die „Kriegserklärung" Microsofts in den so genannten Halloween-Dokumenten seinen schillernden Abschluss findet. Somit ist der Leser gut gerüstet, um dem Autor dabei zu folgen, wie er im Folgenden die vielen Fäden zu einer schlüssigen Theorie webt.

Die beiden zentralen theoretischen Kapitel des Buches widmen sich jeweils der intra- und der interpersonellen Ebene. Weber beantwortet hier zum einen die Frage, warum jemand Zeit und Arbeitskraft in eine Gemeinschaft investiert ohne erkennbare finanzielle Vergütung. Und zum anderen löst er das Rätsel der Koordination und wie Millionen Einzelleistungen sich meist (nicht immer, aber doch signifikant häufig) zu einem funktionierenden Ganzen fügen. Hier kombiniert Weber eine ganze Reihe von Studien und Theorien, die das Thema direkt behandelt haben. Aber er zieht ebenso Werke aus Soziologie, Politologie, Ökonometrie und Philosophie zu Rate.

Sodann widmet Weber das siebte Kapitel der Frage nach der Wirtschaftlichkeit von Free Software. Er durchleuchtet erfolgreiche wie gescheiterte Geschäftsmodelle der neueren Zeit und extrapoliert Basistypen, sozusagen Blaupausen für zukünftige Unternehmungen, aus den verallgemeinerbaren Aspekten. Über den Bereich der Wirtschaft und Softwareentwicklung hinaus schaut Weber im abschließenden achten Kapitel. Denn er sieht in der OSS-Bewegung bei weitem kein auf das Internet oder den Computer begrenztes Phänomen, sondern nahezu einen alternativen Gesellschaftsentwurf, der ein grundlegend neues und von unserer gängigen Auffassung verschiedenes Verständnis von Eigentum beinhaltet. Ein Verständnis, das neu und beachtenswert ist auf Grund seiner positiven Konsequenzen für die Softwareentwicklung, aber keineswegs gängige Muster ersetzen soll. Weber plädiert im Gegensatz stets für die Koexistenz und die gegenseitige Ergänzung verschiedener Ansätze.

Hier liegt auch die große Stärke von Webers Buch, womit er außerdem eine generelle Tugend fortsetzt, die sich in vielen wissenschaftlichen Arbeiten angelsächsischen Ursprungs zeigt. Es ist die Behutsamkeit, mit der die eigene Meinung einge-

bracht und vertreten wird. Weber vermeidet absolute Aussagen, was keineswegs als Schwäche zu verstehen ist, sondern es hilft gerade bei einem Thema, das emotional stark aufgeladen ist und die Industrie in ihren Grundfesten erschüttert. Der Ton der betroffenen Parteien in der Auseinandersetzung um das bessere Modell, Closed Source oder Open Source, bedient sich längst der Kriegslyrik. Doch Weber versteht es, Stärken und Schwächen beider Welten anzusprechen, wie auch die Brisanz des Themas anschaulich zu machen.

Außerdem gereicht es zu Webers Vorteil, kein Informatiker zu sein. Sein Blick auf die Dinge ist der umfassendste, den ich kennengelernt habe. Man braucht zudem kein Informatiker zu sein, um dieses im Internet geborene Phänomen zu verstehen, das er erklärt. Der Aufbau vieler Spannungsbögen ist dialektisch. Weber formuliert die Herausforderung in anschaulichen Fragestellungen, um dann Punkt für Punkt und gut nachvollziehbar die entsprechenden Antworten zu liefern. Die Bezüge zwischen den Kapiteln sind zahlreich, gerade zwischen den beiden erklärenden Teilen fünf und sechs. Da fügen sich die vielen einzelnen Beweise zu einer geschlossenen Theorie, aus der nun ihrerseits Schlüsse gezogen werden können. Obschon Weber im letzten Kapitel dabei etwas über das Ziel hinausschießt, wenn er auf die zwischen supranationalen Institutionen, Regierungen und NGOs zerklüftete Landschaft der Weltpolitik zu sprechen kommt. So hat er doch starke Argumente dafür, dass es sich beim betrachteten Gegenstand nicht allein um einen neuen Softwareentwicklungsprozess handelt.

Das Phänomen Open Source wirft Fragen auf zum Verständnis, was Besitz bedeutet, so stellt es Weber bereits im ersten Satz seines Buches fest. Und das kann im Informationszeitalter Konsequenzen haben, die weit über die Informatik hinausreichen. Wenn der Leser eher an letzteren Schlüssen interessiert ist, kann er die geschichtlichen Kapitel problemlos überspringen. Alle anderen erstaunt Weber mit seiner Detailkenntnis und einer fesselnden Retrospektive, die seine Schlüsse zusätzlich fundieren. Er hat mit seinem Buch das neue Standardwerk zu Open Source und der Informationsgesellschaft vorgelegt.

Kapitel 2

Ökonomie

Einleitung

ANDREAS JOHN

Mit dem Zusammenbruch der New Economy im Jahr 2001 gerieten auch viele Unternehmen, die ihr Geschäftsfeld rund um Open Source angesiedelt hatten, ins Stürzen und Taumeln. Innominate etwa, der damals größte Linux-Dienstleister Europas, hatte noch zuversichtlich mit ganzseitigen Anzeigen im „Spiegel" geworben, kurz bevor er von der Bildfläche verschwand. War es der Sog des niedergehenden Neuen Marktes, die Kehrseite des schnellen Wachstums mittels Risikokapital? Oder lagen die Ursachen für die zahlreichen Insolvenzen in der Grundidee des Open Source begründet – handelte es sich bei der Open-Source-Bewegung lediglich um eine Modeerscheinung, eine Eintagsfliege, ohne langfristige wirtschaftliche Relevanz und Erfolg? Die Tatsachen widersprechen einer derartigen Einschätzung. Das Schlagwort Open Source hallt ungebrochen durch die Medien. In regelmäßigen Abständen erklären Regierungen und Verwaltungen ihre Abkehr von Microsofts Betriebssystem hin zu Open-Source-Systemen. Kosteneinsparungen sind dabei nur ein Motiv. Gerade die letzten großen Virenattacken machten Probleme der Sicherheit und der Abhängigkeit vom Monopolisten aus Redmond nur allzu deutlich. Microsoft versucht derartige Bedenken durch Preisnachlässe und Zugeständnisse in Form der Shared-Source-Strategie[1] auszuräumen. Nicht immer erfolgreich, wie das Beispiel München oder auch das des Bundes- und Versicherungsgerichtes der Schweiz zeigt. Und auch die Unternehmen sind wieder im Kommen. Laut einer Studie des deutschen Marktforschungsunternehmens Soreon wächst der Unternehmensmarkt für Open-Source-Software von 131 Millionen Euro in 2003 auf 307 Millionen Euro in 2007[2]. Johannes Loxen vom Linux-Verband spricht von einer Rückkehr auf die Gewinnerseite, und Bernhard Reiter von der Free Software Foundation Europe sieht die nicht-proprietäre Welt gar als „relativ unabhängig von der Wirtschaftslage"[3].

Die ganzseitigen Anzeigen sind seltener geworden. Dafür haben viele neue Magazine, Zeitschriften und Portale um das Thema Open Source das Licht der Welt erblickt. Fernab des Hypes ist Open Source auf Unternehmensebene realistischer

---

1 Microsoft Shared Source Initiative, http://www.microsoft.com/resources/sharedsource/.

2 Heise online News, 2.7.2003: Studie: Deutscher Open-Source-Markt gedeiht http://www.heise.de/newsticker/data/ola-02.07.03-001/ (zuletzt besucht: 20.01.2004).

3 Krempl, Stefan (2002): Ausgesourct? Freie und Open Source Software nach dem Hype, c't 26/2002

geworden und hat sich stabilisiert. Und das, obwohl es nach gängigen Wirtschaftsmodellen ein ökonomisches Paradox darstellt: Denn der rational handelnde Mensch, an dem sich die klassischen Wirtschaftswissenschaften orientieren, handelt stets mit dem Ziel der Maximierung von Profit oder Nutzen. Auf den ersten Blick will der typische Open-Source-Entwickler, der Zeit und Geld in die Erstellung freier oder zumindest quelloffener Software investiert, in dieses Menschenbild nicht passen. Über die Motivation der Entwickler ist deshalb lange spekuliert worden. Altruistische und ideologische Motive standen zunächst im Fokus dieser Diskussion. Diese beiden Motive waren für Außenstehende ohne detaillierte Rückfragen akzeptabel, und auch die restriktiven Bedingungen der GNU Public Licence (GPL) deuten oberflächlich betrachtet in eine ideologische Richtung. Umfragen machten diesen Spekulationen ein Ende. Altruismus und Ideologie sind nur zwei von vielen möglichen Motivationen, die Entwickler und Tester dazu bewegen, in der Open-Source-Bewegung aktiv mitzuwirken.

Die Motivation der Entwickler als Schlüssel für eine Klärung des Phänomens Open Source war und ist Gegenstand vieler wissenschaftlicher Untersuchungen und Umfragen. Mit 5478 Teilnehmern wurde 2001 in einem Projekt des Institutes Informatik und Gesellschaft der TU Berlin die bisher größte Umfrage unter Softwareentwicklern durchgeführt[4].

Das Forschungsprojekt „Fun and Software Development" (FASD) des Lehrstuhles Unternehmenführung und -politik der Universität Zürich hat seinen Fokus auf die Untersuchung der intrinsischen Motivation Spaß und ihre Bedeutung für den Erfolg einer Softwareentwicklung gelegt. In seinem Artikel „Alles aus Spaß?" beschreibt der Leiter des Projektes *Benno Luthiger* die Bedeutung der Motivationen für die Klärung des Paradoxons Open Source, gibt einen Überblick über ausgewählte intrinsische und extrinsische Motivationen der Softwareentwickler und stellt den Gegenstand des FASD-Projektes[5] vor.

Auch auf Unternehmensebene erfordert der Umgang mit Open Source eine veränderte Sichtweise. Innovation und Marktposition können bei einem quelloffenen Softwareprodukt nicht in gleichem Maße geschützt werden, wie das bei der Entwicklung von kommerzieller proprietärer Software oder auch Shareware der Fall ist. Denn Open-Source-Software ist ein öffentliches Gut, das jeder entsprechend den gewählten Lizenzmodellen einsehen, verwenden, modifizieren, kopieren und verkaufen kann. Durch die Freigabe des Quellcodes wird der Wert der Software als handelbare Ware bedroht, ihr wirtschaftlicher Wert als Werkzeug bleibt unangetastet. Der hohe Gebrauchswert z.B. durch größere Zuverlässigkeit, mehr Sicherheit und offene Schnittstellen ist es auch, der viele große Unternehmen bewegt, Teile ihrer IT-Struktur mit Open-Source-Software zu realisieren. Die Einsparung von Lizenzgebühren spielt nur eine geringe Rolle, da die Umstellung auf Open Source in den wenigsten Fällen kostenlos ist. Auch wenn beispielsweise der Preis einer Linux-Distribution weit niedriger ausfällt als die kommerzieller proprietärer Betriebssysteme, entstehen gerade bei größeren Unternehmen nicht unerhebliche Migrations-

4 Robles, G., Scheider, H., Tretkowski, I. & Weber, N. (2001): Who Is Doing It? A research on Libre Software developers, Fachgebiet für Informatik und Gesellschaft, TU Berlin.

5 FASD-Projekt, http://www.ifbf.unizh.ch/webFuehrung/blprojects/FASD/.

und Schulungskosten. Auch das Vorzeigeprojekt Münchener Stadtverwaltung hat nach anfänglicher Euphorie bei der Migration auf Open-Source-Software mit ganz realen technischen und organisatorischen Problemen und auch persönlichen Widerständen zu kämpfen.[6]

Die Gefahren von Open-Source-Software werden von einer durch Microsoft in Auftrag gegebenen Studie des Münsteraner Universitätsablegers MICE ausführlich beleuchtet.[7] Darin werden Open-Source-Software erhebliche ökonomische Defizite durch Innovationsfeindlichkeit vorgeworfen. Die unentgeltliche Weitergabe der Software, so die Studie, verhindere bewusst das Entstehen eines preisgesteuerten Softwaremarktes und damit das Setzen von Innovationsanreizen. Weiterhin heißt es in der Studie, werde bei Open Source an Nutzerinteressen vorbeientwickelt. Demgegenüber steht die Tatsache, dass Open-Source-Software bereits 2002 bei 43 % innerhalb der FLOSS-Studie untersuchten Unternehmen und öffenlichen Institutionen im Einsatz war. 25% dieser Einrichtungen schrieben der Software eine hohe bis sehr hohe Bedeutung für ihre Infrastruktur zu. Die Kritiker werfen der Studie falsche Annahmen und die Verhedderung in eigener Ideologie vor. Als Entscheidungsvorlage für die notwendige individuelle Prüfung eines Einsatzes von Open Source im Unternehmen erscheint sie deshalb ungeeignet.

Diese Lücke schließt der Beitrag „Stärken und Schwächen freier und Open-Source-Software im Unternehmen“ von *Thomas Wieland*, Professor für Telematik, Mobile Computing und Computergrafik an der Fachhochschule Coburg und Gründungschefredakteur der Zeitschrift „Linux Enterprise“. Vorteile und Risiken, die für oder wider einen Einsatz oder eine Beteiligung an der Entwicklung von Open Source sprechen, werden vom Autor gleichermaßen und unter objektiven Gesichtspunkten beleuchtet.

Durch die Konzentration auf den Gebrauchswert lässt sich mit Open-Source-Software auch Geld verdienen. Eric S. Raymond, Namensgeber des Begriffes „Open-Source-Software“ und Autor des Artikels „The Magic Cauldron“, spricht Software sogar generell den Anspruch auf einen Marktwert ab, da sich der Wert einer Software nicht an den Entwicklungskosten orientiert und ohne Support und Updates schnell gegen null sinkt, wenn das Softwareunternehmen Bankrott geht. Raymond zufolge sind Softwareunternehmen Dienstleister und keine Güterindustrie. Der Wert der Software definiert sich daher über den wirtschaftlichen Wert als Werkzeug und das Angebot an Erweiterungen, Updates und Folgeprojekten.

Der Dienstleistungsbereich ist es auch, der das Geschäftsfeld der meisten Unternehmen rund um Open Source bildet. Diese und weitere Möglichkeiten für Open Source als Grundlage von Geschäftsmodellen behandelt *Raphael Leiteritz* in seinem Artikel „Open-Source-Geschäftsmodelle“. Raphael Leiteritz ist Gründer des bereits zu Beginn der Einleitung erwähnten Unternehmens Innominate und ehemaliger Wissenschaftler an der TU Berlin. In seinem Artikel beschreibt er sowohl Dienstleistungs- als auch Produkt- und Mediatoren-Geschäftsmodelle für Open-

[6] Heise online News, 9.1.2004: Probleme für Linux in München http://www.heise.de/newsticker/data/anw-09.01.04-000/ (zuletzt besucht: 20.1.2004).

[7] Krempl, Stefan (2004): Ja zum Monopol, Microsoft, Open Source und die Tücken der Volkswirtschaft, „c't“ 1/2004, http://www.heise.de/ct/04/01/045/.

Source-Software. Untersucht werden Fragen der Marktpositionierung, Gewinnmuster, Recourcenfokus, strategische Absicherung sowie Organisation, Mitarbeiter und Kultur der Unternehmensmodelle.

Der Erfolg von Open-Source-Software hat nicht nur im IT-Bereich hellhörig gemacht. Der vordergründige Widerspruch zu klassischen ökonomischen Modellen forderte die Entwicklung neuer Erklärungsversuche für die sich aufdrängenden Fragen heraus: Unterliegt das öffentliche Gut Open Source nach einem anfänglichen Boom in wenigen Jahren durch Unterversorgung und Übernutzung der „Tragedy of the Common"?[8] Welche Auswirkungen hat Open Source auf das bestehende Wirtschaftssystem, auf Wettbewerb und Innovation?

Für die Klärung der Fragen wurden viele Modelle bemüht: Das Gleichnis des Basars von Eric S. Raymond ist wohl das bekannteste. Aber auch das Kochtopfmodell von Rishab Aiyer Ghosh[9] sowie verschiedene Modelle über Aufmerksamkeits- und Geschenkökonomien beleuchten das Phänomen Open Source aus unterschiedlichen Perspektiven.

Die Suche nach einem umfassenden erklärenden Modell ist vor allem für die Untersuchung der Übertragbarkeit des Open-Source-Entwicklungsmodells auf andere wirtschaftliche Bereiche von Belang. In der Tat sind die Aspekte des offenen Wissens nicht neu, sie sind vielmehr Voraussetzung für wissenschaftliches Arbeiten an Universitäten überhaupt. Auch das freiwillige Bereitstellen von Ressourcen gibt es schon seit längerem z.B. bei der Abgabe freier Rechenleistung für die Analyse von Radiowellen aus dem Weltall innerhalb des SETI-Projektes[10]. Und Konzepte des peer review werden auch in klassischen IT-Unternehmen bei der Software-Entwicklung angewandt, wenn auch nicht in dem Maße, wie dies bei frei zugänglichem Quellcode und Netzwerkeffekten durch das Medium Internet der Fall ist.

Der Artikel „Open-Source-Software-Produktion: Ein neues Innovationsmodell?" von *Margit Osterloh, Sandra Rota und Bernhard Kuster* des Institutes für Betriebswirtschaftslehre der Universität Zürich untersucht, unter welchen Bedingungen das Open-Source-Modell generalisiert werden kann und wie motivationale, situative und institutionelle Faktoren beschaffen sein müssen, damit Innovation auch ohne geistiges Eigentum und zentrale Autorität des Unternehmers realisiert werden kann.

Die vorgestellten nun folgenden Artikel sollen helfen, ein Problembewusstsein für die Ökonomie von Open Source fern von Vorurteilen, Hype und Ideologien zu entwickeln. Denn der langfristige Erfolg des Entwicklungsmodells hängt maßgeblich von der Klärung der ökonomischen Fragestellungen ab.

---

8 G. Hardin (1968): „The tragedy of the commons", Science 162, S. 1243–1248.

9 R.A. Ghosh (1998): „Cooking pot markets: an economic model for the trade in free goods and services on the Internet.", Firstmonday volume 3, number 3, http://www.firstmonday.org/issues/issue3_3/ghosh/index.html.

10 SETI@home, http://setiathome.ssl.berkeley.edu/.

Literatur

Fehr, E. / Rockenback, B. (2003): *Detrimental effects of sanctions on human altruism*, Nature Publishing Group

International Institute of Economics, University of Maastricht and Berlecon Research (2002): *FLOSS – Free/Libre and Open Source Software: Survey and Study* online http://www.infonomics.nl/FLOSS/index.htm

Johnson, Justin Pappas (2001): *The Economics of Open Source Software*, online http://www.polter.net/~bob/files/johnsonopensource.pdf (zuletzt besucht: 20.1.2004)

Goldhaber, Michael. H. (1997): *The Attention Economy of the Net*, online http://www.firstmonday.dk/issues/issue2_4/goldhaber/ (zuletzt besucht: 20.1.2004)

Ghosh, Rishab Aiyer (1999): *Cooking-pot markets: an economic model for „free" resources on the Internet*

Heise Online News, 9.1.2004: *Probleme für Linux in München*, online http://www.heise.de/newsticker/data/anw-09.01.04-000/ (zuletzt besucht: 20.1.2004)

Heise Online News, 2.7.2003: *Studie: Deutscher Open-Source-Markt gedeiht*, online http://www.heise.de/newsticker/data/ola-02.07.03-001/ (zuletzt besucht: 20.1.2004)

Krempl, Stefan (2004): *Ja zum Monopol, Microsoft, Open Source und die Tücken der Volkswirtschaft*, „c't" 1/2004, http://www.heise.de/ct/04/01/045/

Krempl, Stefan (2002): *Ausgesourct? Freie und Open Source Software nach dem Hype*, „c't" 26/2002.

Luthiger, Benno (2002): *Open Source als Gegenstand wissenschaftlicher Forschung*, online http://www.ch-open.ch/html/events/obl_vortraege/Luthiger_OBL_Okt02.pdf (zuletzt besucht: 20.1.2004)

Osterloh M., Rota S., Kuster B. (2002): *Open Source Software Production: Climbing on the Shoulders of Giants*

Raymond, Eric S. (1999): *The Cathedral and the Bazaar*, online http://www.catb.org/~esr/writings/cathedral-bazaar/ (zuletzt besucht: 20.1.2004).

Robles, G. / Scheider, H. / Tretkowski, I. / Weber, N. (2001): *Who Is Doing It? A research on Libre Software developers*, Fachgebiet für Informatik und Gesellschaft, TU Berlin, online http://widi.berlios.de/paper/study.pdf

Sebald, Gerd (2002): *Geschenkökonomie im luftleeren Raum*, online http://parapluie.de/archiv/cyberkultur/opensource (zuletzt besucht: 20.1.2004)

Alles aus Spaß?
Zur Motivation von Open-Source-Entwicklern

Benno Luthiger

Ein PC-Benutzer geht normalerweise davon aus, dass gute Software wertvoll ist und deshalb etwas kostet. Es ist nachvollziehbar, dass solche Benutzer erstaunt sind, wenn sie das erste Mal mit Open-Source-Software in Kontakt kommen, erst recht, wenn die Software von überzeugender Qualität ist. Wie kommt es, dass Personen solche Software schreiben und freigeben, wenn sie mit dieser Software viel Geld verdienen könnten?

In diesem Artikel wird im ersten Kapitel gezeigt, wie das Paradox des Open-Source-Phänomens aufgelöst werden kann, indem die geeignete Perspektive eingenommen wird. Nicht die Softwarefirmen, sondern die einzelnen Softwareentwickler sind die Aktoren, welche die Existenz von Open-Source-Software erklärbar machen.

Im folgenden Kapitel wird dargelegt, dass eine ganze Reihe von Motivationen die Open-Source-Entwickler zu ihrer Tätigkeit anspornen. Die immer zahlreicheren empirischen Untersuchungen zu Open-Source-Software bestätigen die Existenz dieser Motivationen weitgehend. Was allerdings noch fehlt, sind Abschätzungen über die Relevanz dieser verschiedenen Beweggründe.

Im dritten Kapitel wird das Forschungsprojekt „Fun and Software Development" (FASD) vorgestellt. Dieses Forschungsprojekt ist Teil der Promotion des Autors am Institut für betriebswirtschaftliche Forschung der Universität Zürich. Ziel der FASD-Studie ist es, den Anteil, den Spaß bei der Entstehung von Open-Source-Software spielt, möglichst genau zu bestimmen. Zu diesem Zweck werden mit einem Online-Fragebogen mindestens 2000 Open-Source-Entwickler sowie ca. 1000 Entwickler in kommerziellen Softwarefirmen befragt. Die Umfrage wird im ersten Halbjahr 2004 durchgeführt. Erste Ergebnisse werden Ende 2004 erwartet.

1. Benutzer-Programmierer als Open-Source-Aktoren

Mit Software kann man viel Geld verdienen. Sowohl der Einsatz wie auch die Erzeugung von Software machen erhebliche Gewinne möglich. Dazu kommt, dass Computer und softwaregesteuerte Apparate den Alltag, vor allem den geschäftlichen, durchdringen. Diese Erfahrungen machen es naheliegend, Phänomene im Zusammenhang mit Software in erster Linie durch die wirtschaftliche Brille zu betrachten.

In einer solchen wirtschaftlich geprägten Sichtweise erscheint das Open-Source-Phänomen offensichtlich als Paradox: Warum engagieren sich Leute, um ein qualitativ hochstehendes Produkt zu erzeugen, wenn dieses in der Folge frei zur Verfü-

gung gestellt wird? Warum verschenkt die eine Person ein wertvolles Gut, wo doch die andere damit viel Geld verdient?

Eine Möglichkeit, dieses Paradox aufzulösen, ergibt sich aus der Verschiebung des Blickwinkels. Statt, wie in der ökonomisch geprägten Sichtweise naheliegend, die Softwarefirmen als die relevanten Aktoren zu betrachten, werden die Software-Benutzer-Programmierer ins Zentrum der Untersuchung gerückt. Im Zentrum der Analyse des Open-Source-Phänomens stehen dann die ‚Prosumer' (siehe Toffler 1980), d. h. Benutzer, welche die Software an ihre Bedürfnisse anpassen und weiterentwickeln. Wie von Hippel und von Krogh (2002) gezeigt haben, ist mit einem solchen Perspektivenwechsel ein wesentlicher Erkenntnisgewinn möglich.

Das heißt aber nicht, dass Prosumer weniger rational handeln als Softwarefirmen. Für beide Arten von Aktoren gilt, dass Software dann freigegeben wird, wenn der Nutzen einer Offenlegung des Quellcodes größer ist als die Kosten einer solchen Handlung. Nur präsentiert sich das Umfeld und damit auch die Rechnung für die Softwarefirmen anders als für die Prosumer. Softwarefirmen befinden sich in einem ausgeprägten Konkurrenzverhältnis zueinander. Für sie ist eine Freigabe des Quellcodes mit hohen Opportunitätskosten verbunden: Mit einer Offenlegung des Quellcodes geben sie ein Geschäftsgeheimnis preis und verspielen dadurch einen Wettbewerbsvorteil.

Für die Software-Benutzer-Programmierer sieht die Lage ganz anders aus. Das Verhältnis der Prosumer untereinander ist durch bloß geringe Rivalitätsbedingungen bestimmt. Zusätzlich gilt, dass dank dem Internet auch die direkten Kosten gering sind: die Verbreitung des Quellcodes verursacht nur minimale Kosten.

Doch der geringe Aufwand auf der Kostenseite kann das Verhalten der Software-Benutzer-Programmierer noch nicht erklären. Für rationale Aktoren wäre es immer noch sinnvoller, als Trittbrettfahrer am Open-Source-Phänomen teilzunehmen, d. h. ohne eigene Leistungen von den Arbeiten anderer zu profitieren, statt aktiv an der Erzeugung solcher Software mitzuwirken. Für einen Beitragsleister muss in irgendeiner Form ein selektiver Vorteil existieren, ein Nutzen, den nur jene Personen genießen können, die sich engagieren. Allerdings gilt in Situationen, wo die Kosten einer Freigabe gering sind (sog. „low cost"-Situationen), dass solche selektiven Vorteile nicht mehr groß zu sein brauchen, damit für die Benutzer-Programmierer ein Engagement für Open-Source-Software vorteilhaft erscheint.

Wenn es nun gelingt, solche selektiven Vorteile, von welchen nur die Prosumer, nicht aber die Trittbrettfahrer profitieren können, zu identifizieren und zu quantifizieren, so ist das eingangs beschriebene Paradox im Wesentlichen aufgelöst.

2. Motivationen

Welche selektiven Vorteile könnten also für Benutzer-Programmierer relevant sein und diese zu einem Engagement für Open-Source-Software motivieren?

In der wissenschaftlichen Literatur über Open Source finden sich etliche Analysen, die sich mit diesem Thema auseinandersetzen. Auf Grund dieser Untersuchungen konnte ein ganzes Bündel von Anreizen identifiziert werden. Die mittlerweile vorliegenden empirischen Studien über das Open-Source-Phänomen (Robles u.a.

2001, Jorgensen 2001, Hars und Ou 2001, Dempsey u.a. 2002, Ghosh u.a. 2002, Krishnamurthy 2002, Bonaccorsi und Rossi 2003, Hertel u.a. 2003, Healy und Schussman 2003, Lakhani und Wolf 2003) konnten die Existenz dieser unterschiedlichen Motivationen tatsächlich bestätigen (siehe Tabelle 1). In den folgenden Kapiteln werden diese Motivationen genauer erläutert.

2.1. Gebrauch

Die einfachste Begründung dafür, dass sich ein Softwareentwickler für ein Open-Source-Projekt engagiert, ist, dass er die von diesem Projekt erzeugte Software gebrauchen kann. Dieses Motiv entspringt dem Prosumer-Modell: Der Aktor hat ein Problem, welches mit der geeigneten Software gelöst werden kann, und erzeugt die entsprechende Software oder passt eine existierende für seine Bedürfnisse an. In der Studie von Lakhani und Wolf (2003) erhielt dieses Motiv die größte Zustimmung.

2.2. Reputation und Signalproduktion

Raymond hat in seinem letzten Essay „Homesteading the Noosphere“ (2000) aus seiner vielbeachteten „The Cathedral and the Bazaar“-Trilogie beschrieben, wie die Normen und Tabus der Open-Source-Bewegung in Bezug auf den Erwerb von Reputation wirken. Wie Raymond aufzeigt, sind „code forking“, d. h. das Aufsplitten der Code-Basis in inkompatible Versionen, und vor allem das Löschen der Namen der Beitragsleister aus den „credit files“ der Applikationen streng verpönt. Die Sensibilität der Open-Source-Community gegenüber diesen Normen macht Sinn im Hinblick auf den Aufbau von Reputation, denn ohne diese Normen wäre es viel schwieriger nachzuvollziehen, welcher Beitrag von welcher Person kommt, und damit wäre der Aufbau von Reputation stark erschwert.

Lerner und Tirole (2001) haben diesen Zusammenhang unter dem Aspekt der Signalproduktion analysiert. Gemäß dieser Argumentation ermöglicht es die Offenlegung des Quellcodes zusammen mit den spezifischen Normen der Open-Source-Community wie von Raymond dargestellt, dass die Beiträge der einzelnen Entwickler sehr gut verfolgt werden können. Dies hat zur Folge, dass der Status eines Entwicklers in einem Projekt ein genaues Abbild seiner Reputation ist und diese wiederum sehr transparent die Qualität und Quantität seiner Beiträge reflektiert. Vor diesem Hintergrund besteht nun die Möglichkeit, diese Reputation zu monetarisieren, z. B. durch ein interessantes Arbeitsangebot oder einen verbesserten Zugang zu Risikokapital. Der Arbeits- bzw. Risikokapitalmarkt ist primär an Talent und Leistung und nicht an Reputation interessiert. Das Talent einer Person ist aber für Personen, die nicht mit dem Fachgebiet vertraut sind, schwer einzuschätzen. Wenn allerdings die Reputation einer Person ein gültiger Indikator für deren Talent ist, kann Reputation im oben beschriebenen Sinn als Signal wirken: Aus dem Wissen über ein Open-Source-Projekt und der Position einer Person in diesem Projekt kann z.B. ein potentieller Arbeitgeber gültige Rückschlüsse auf das Talent der Person ziehen. Signalproduktion wirkt am stärksten, wenn die technische Herausforderung groß ist, wenn die relevante Öffentlichkeit (d.h. die Peergruppe) technisch erfahren ist, zwischen guten und herausragenden Leistungen unterscheiden sowie Leistung und Können

wertschätzen kann (Weber 2000, Franck und Jungwirth 2001). Diese Bedingungen sind im Falle von Open Source in hohem Maß gegeben.

2.3. Identifikation mit der Gruppe

Ist eine Person gut in eine Gruppe eingebunden und kann sie sich stark mit den Gruppenzielen identifizieren, so kann häufig ein großes Engagement dieser Person für die Gruppenziele beobachtet werden (Kollock und Smith 1999). Hertel u.a. (2002) untersuchten in einer empirischen Studie unter Linux-Entwicklern, ob dieses Phänomen auch im Open-Source-Bereich von Bedeutung ist. Sie haben die Entwickler über verschiedene Aspekte ihrer Tätigkeit befragt und diese Angaben mit den Werten über das Engagement korreliert. Mit statistischen Methoden gelang es ihnen tatsächlich nachzuweisen, dass ein signifikanter Anteil des Engagements der Entwickler durch ihre Identifikation mit ihrer Entwicklergemeinschaft erklärt werden kann.

Dieses Resultat ist von Lakhani und Wolf (2003) bestätigt worden. In ihrem Hacker-Survey untersuchten Lakhani und Wolf, wie sich die Identifikation mit der Projektgruppe auf das zeitliche Engagement auswirkt, und konnten dabei einen signifikant positiven Effekt feststellen.

2.4. Lernen

Der Einsatz für ein Open-Source-Projekt kann auch unter dem Aspekt des Lernens erklärt werden. Der Einsatz von neuster Technologie zur Bewältigung eines komplexen Problems ist für alle Beteiligten mit einem Gewinn an Erfahrung verbunden. Der Wunsch, seine Fähigkeiten als Softwareentwickler zu verbessern, taucht sowohl in der FOSS Studie von Ghosh u.a. (2002) wie auch im Hacker-Survey von Lakhani und Wolf (2003) und in der Untersuchung von Hars und Ou (2001) mit hohen Zustimmungsraten auf.

2.5. Altruismus

Ein Engagement für Open Source kann auch begründet werden mit einem Gefühl für das Wahre und Richtige, z.B. dass die Freiheit der Menschen mit quelloffener Software zusammenhängt und dieses Gut gefährdet ist durch kommerzielle und proprietäre Softwareanbieter. Dieser Standpunkt wird pointiert von Stallman und seiner Free Software Foundation (FSF) vertreten.

Um eine ähnliche Motivation handelt es sich, wenn ein Programmierer an einem Open-Source-Projekt beteiligt ist, weil er selbst viele Open-Source-Produkte verwendet und nun die Verpflichtung spürt, selbst etwas für die Open-Source-Bewegung zu tun.

Während das erste Motiv auf ein abstraktes Gefühl und eine ideologische Haltung verweist, referiert das zweite Motiv auf ein Gefühl des gerechten Gebens und Nehmens, einer allgemeinen Reziprozität im Verhältnis des Individuums zu einer Gruppe. Ökonomisch können die auf diese Weise motivierten Beiträge als Spenden interpretiert werden. Ob der Beitrag geleistet wird oder entfällt, hat für den Programmierer keine nachvollziehbaren Auswirkungen. Das Open-Source-Projekt, in

welchem sich der Spender engagiert, profitiert aber von den Beiträgen dieser Person.

Gemäß der Studie von Lakhani und Wolf (2003) bezeichneten ungefähr ein Drittel der befragten Open-Source-Programmierer diese Motive als relevant für ihr Engagement. Interessant ist, dass Lakhani und Wolf mit einer Cluster-Analyse zeigen konnten, dass das ideologisch fundierte Motiv, Software müsse offen sein, tatsächlich sehr nahe beim Motiv liegt, welches das Engagement aus dem Gefühl der Reziprozität heraus begründet.

2.6. Spaß

Seit es Computer gibt, üben diese Geräte, vor allem auf Männer, eine große Faszination aus. Die Schilderungen der Befriedigung und des Spaßes, den die Programmierer bei ihrer Tätigkeit empfinden, ziehen sich wie ein roter Faden durch die Lebensgeschichten von Computerexperten und Hackern. Brooks, der als Computerwissenschaftler und Softwarepionier bei IBM wirkte, beschreibt diesen Sachverhalt wie folgt: „Programming [then] is fun because it gratifies creative longings built deep within us and delights sensibilities we have in common with all men." (1995, S. 8). Linus Torvalds betitelt seinen Rückblick über die Entstehung von Linux schlicht mit „Just for FUN" (2001).

Doch die Tatsache, dass Programmieren Spaß macht, reicht noch nicht aus als Erklärung dafür, das Open-Source-Software programmiert wird. Auch Spaßsucher sind primär rational denkende und handelnde Personen. Wenn Programmieren Spaß macht, dann ist es immer noch besser, mit dieser Tätigkeit Geld zu verdienen, als das Resultat der Arbeit zu verschenken. Um die Bedeutung von Spaß wirklich zu verstehen, muss noch gezeigt werden, dass und wie sich Programmieren in einem Open-Source-Projekt unterscheidet von der Tätigkeit eines Softwareentwicklers in einem kommerziellen Unternehmen.

Solche Unterschiede können tatsächlich identifiziert werden:

- Der Projektmanager in einem kommerziellen Umfeld ist üblicherweise nicht diejenige Person, welche die Vision über die Applikation entwickelt hat und pflegt.
- Der Projekteigentümer in einem Open-Source-Projekt hat keine formale Autorität.
- Die Programmierer in einem Open-Source-Projekt können üblicherweise nicht über direkte monetäre Anreize zu einem Engagement oder zur Steigerung ihres Engagements bewegt werden.
- Ein Open-Source-Projekt hat üblicherweise keine Abgabetermine.

Solche Differenzen machen es nachvollziehbar, dass das Open-Source-Entwicklungsmodell mehr Spaß zulässt, als unter kommerziellen Produktionsbedingungen möglich ist.

Die Bedeutung von Spaß als Motivation für ein Engagement im Open-Source-Bereich wird von den bisherigen empirischen Studien weitgehend bestätigt. Beispielsweise ergaben die Fragen nach Kreativität und Flow in der Studie von Lakhani und Wolf (2003) außerordentlich hohe Zustimmungsraten von 61% respektive 73%.

2.7. Verträglichkeit der unterschiedlichen Motivationstypen

Osterloh u. a. (2002) unterteilen die möglichen Motivationen auf der ersten Stufe in intrinsische und extrinsische Motivationen (z.B. Signalproduktion, Gebrauch). Innerhalb der intrinsischen Motivationen unterscheiden sie wiederum in solche, die auf Freude basieren („enjoyment based", z.B. Spaß) und solche, die auf Verpflichtungen gegenüber der Gemeinschaft basieren („obligation"/„community based", z.B. Gruppenidentifikation).

Eine weitere Möglichkeit, die unterschiedlichen Motive zu kategorisieren, haben Franck und Jungwirth (2002b) vorgeschlagen. Sie unterscheiden zuerst zwischen Spendern und Rentensuchern. Als Spender werden diejenigen Beitragsleister bezeichnet, die sich aus ideellen oder altruistischen Gründen für Open Source engagieren. Rentensucher dagegen stellen sich die Frage, ob sich das Engagement für sie rentiert. Diese zweite Kategorie kann nochmals unterteilt werden in Investoren und Konsumenten. Für einen Entwickler, der sich im Sinne einer Investition in einem Open-Source-Projekt engagiert, zahlt sich das Engagement erst im Verlauf der Zeit z.B. als gestiegene Reputation oder als Folge der größeren Fähigkeiten aus. Ein Entwickler, der sich aus Freude am Programmieren beteiligt oder die Software zur Lösung eines Problems braucht, zieht dagegen einen unmittelbaren Nutzen aus seiner Tätigkeit.

Wesentlich ist, dass sich die unterschiedlichen Motive nicht gegenseitig ausschließen, sondern, im Gegenteil, ergänzen. Dieser Umstand liegt im Wesen der Open-Source-Lizenz begründet. Eine Open-Source-konforme Lizenz unterstellt die Software einer „Non Distribution Constraint" (siehe dazu Hansmann 1980). Damit ist sichergestellt, dass die Beiträge der Spender nicht entgegen deren Intention proprietär vereinnahmt werden können. Aber auch für Rentensucher, die im Hinblick auf ihre Reputation investieren, ist die Open-Source-Lizenzbedingung von Bedeutung. Erst die Garantie, dass der Quellcode offen bleibt, ermöglicht den transparenten Nachweis der Leistung und fundiert damit die Reputation der Beitragsleistenden.

Weiter ist es für Rentensucher, die im Hinblick auf die Signalproduktion in ihre Reputation investieren, rational, in reife Projekte zu investieren, welche schon einen ersten Markttest bestanden haben oder kurz davor sind, einen gewissen Marktanteil zu erobern. Mit neuen Open-Source-Projekten dagegen können noch keine relevanten Signale produziert werden. Durch Reputation extrinsisch motivierte Entwickler brauchen also Spender oder „Fun seeker", damit ein Open-Source-Projekt überhaupt entsteht. Umgekehrt ist die Teilnahme von profilsuchenden Investoren auch für Spender vorteilhaft. Wer Signale produzieren will, ist eher bereit, Kundenwünsche aufzunehmen und diese in das Projekt zu integrieren. Das Eingehen auf Kundenbedürfnisse ist aber für den langfristigen Erfolg eines Softwareprodukts unabdingbar.

Diesen Sachverhalt, dass in der Open-Source-Bewegung Personen mit ganz unterschiedlichen Motivationen beteiligt sind und miteinander kooperieren, konnten Lakhani und Wolf (2003) in ihrer empirischen Studie bestätigen.

3. Das Forschungsprojekt „Fun and Software Development“

Spaß als Motivation für ein Engagement in ein Open-Source-Projekt ist der Gegenstand des Forschungsprojekts „Fun and Software Development“ (FASD). Ziel dieses Forschungsprojekts ist es, eine möglichst genaue quantitative Abschätzung über die Bedeutung dieses Motivationsfaktors zu erzielen.

3.1. FASD-Modell

Im FASD-Forschungsprojekt wird ein Modell benutzt, welches einen Zusammenhang zwischen Spaß und Engagement herstellt. Dieses Modell orientiert sich an der Theorie kompensierender Lohndifferenziale (Lorenz und Wagner 1988, Backes-Gellner u. a. 2001). Diese Theorie erklärt nachweisbare Lohnunterschiede durch das Vorhandensein von nicht-monetären Faktoren. So honorieren z.B. Angestellte größere Flexibilität bei der Arbeitszeitgestaltung durch Zurückhaltung bei Lohnforderungen oder wollen ein größeres Unfallrisiko am Arbeitsplatz durch höheren Lohn entschädigt haben. Angewendet auf die Aufgabe, das Phänomen Open Source zu erklären, besagt das Modell, dass Programmierer für weniger Lohn arbeiten, je mehr Spaß die Tätigkeit bereitet. Im Idealfall der Open-Source-Situation macht das Entwickeln von Software so viel Spaß, dass eine monetäre Entschädigung überhaupt keine Rolle mehr spielt. Wenn im typischen Open-Source-Fall ohne Lohn programmiert wird, so handelt es sich dabei um eine Freizeitbeschäftigung. Eine Tätigkeit in der Freizeit ist aber nur möglich, wenn überhaupt Freizeit verfügbar ist. Wir müssen unser Modell also durch den Faktor „Freizeit“ ergänzen.

Auf Grund dieser Überlegungen besagt das Modell, welches mit der FASD-Studie überprüft werden soll, Folgendes: Je mehr Spaß das Programmieren macht und je mehr Freizeit der Softwareentwickler hat, desto größer ist sein Einsatz beim Entwickeln von Open-Source-Software.

3.2. Flow-Konzept

Zur Ermittlung des Motivationsfaktors „Spaß“ wird für die FASD-Studie das von Csikszentmihalyi (1975) entwickelte Flow-Konzept verwendet. Flow erscheint aus zwei Gründen besonders tauglich, Spaß beim Programmieren zu messen. Einerseits ist das Empfinden von Flow als psychologisches Konstrukt schon relativ gut untersucht worden. Es existieren eine ganze Reihe von empirischen Studien zu Flow und ein entsprechendes Wissen, wie dieses Konstrukt gemessen werden kann. Andererseits scheint speziell die Tätigkeit des Softwareentwickelns gut geeignet, Flow-Empfindungen hervorzurufen. Ein großer Teil der Flow-Studien haben gerade Tätigkeiten mit und an Computern verwendet, um Flow zu erforschen.

Dieser Zusammenhang erstaunt nicht, wenn man die Komponenten betrachtet, aus denen Flow besteht (zitiert aus Rheinberg 1997, S. 143):

- Handlungsanforderungen und Rückmeldungen werden als klar und interpretationsfrei erlebt, sodass man jederzeit und ohne nachzudenken weiß, was jetzt als richtig zu tun ist.
- Man fühlt sich optimal beansprucht und hat trotz hoher Anforderungen das sichere Gefühl, das Geschehen noch unter Kontrolle zu haben.

- Der Handlungsablauf wird als glatt erlebt. Ein Schritt geht flüssig in den nächsten über, als liefe das Geschehen gleitend wie aus einer inneren Logik. (Aus dieser Komponente rührt wohl die Bezeichnung „Flow".)
- Man muss sich nicht willentlich konzentrieren, vielmehr kommt die Konzentration wie von selbst, ganz so wie die Atmung. Es kommt zur Ausblendung aller Kognitionen, die nicht unmittelbar auf die jetzige Ausführungsregulation gerichtet sind.
- Das Zeiterleben ist stark beeinträchtigt; man vergisst die Zeit und weiß nicht, wie lange man schon dabei ist. Stunden vergehen wie Minuten.
- Man erlebt sich selbst nicht mehr abgehoben von der Tätigkeit, man geht vielmehr gänzlich in der eigenen Aktivität auf (sog. „Verschmelzung" von Selbst und Tätigkeit). Es kommt zum Verlust von Reflexivität und Selbstbewusstheit.

In der Idealsituation für einen Softwareentwickler, wenn sich der Programmierer ungestört seiner selbstgewählten Aufgabe widmen kann, wenn er vor einem auf seine Bedürfnisse zugeschnittenen und konfigurierten Computer sitzt, ein Stück seines Projekts implementiert, dieses testet, die Ergebnisse analysiert und auf Grund der Rückmeldungen sein Werk verbessert, seinen neuen Code in die bestehende Software integriert und den nächsten Teil in Angriff nimmt, wenn sich unter seinen Händen ein raffiniertes Stück Software bildet und dieses Schritt für Schritt wächst, an Eleganz und Leistungsfähigkeit gewinnt, in einer solchen Situation sind die Voraussetzungen für das Erleben von Flow optimal.

Eine in Flow-Studien oft verwendete Methode besteht darin, den Probanden eine Reihe von Frage-Items vorzulegen (20 bis 40 Stück), die verschiedene Gefühlsempfindungen beschreiben. Die Probanden können dann auf einer Skala wählen, wie gut die Beschreibungen mit ihren Empfindungen übereinstimmen. Mit Hilfe von Faktoranalysen können diese Antworten in der Folge statistisch ausgewertet werden. Solche Analysen zeigen übereinstimmend, dass Flow als Konstrukt im Wesentlichen aus den Komponenten „Flow-Erleben", „Eindeutigkeit der Aufgabe" und „Erlebte Leichtigkeit" besteht (z.B. Remy 2000).

3.3. Online-Umfrage

Die in der FASD-Studie gewählte Methode orientiert sich an der oben beschriebenen Vorgehensweise. Die Online-Umfrage des FASD-Projekts besteht aus rund 30 Items, welche sich auf die Bestimmung von Flow beziehen. In diesem Teil werden die Probanden gefragt, wie häufig eine Empfindung wie beispielsweise „Alles scheint wie von selbst zu laufen" oder „Meine Aufmerksamkeit ist völlig auf die Handlung gelenkt" zutrifft, wenn sie Software entwickeln. Die Teilnehmer beantworten diese Fragen, indem sie einen Wert auf einer sechspunktigen Likertskala zwischen „nie" und „immer" wählen.

In den weiteren Teilen des Fragebogens werden die Probanden gefragt, wie sie ihr zukünftiges Engagement für Open-Source-Projekte einschätzen und welchen Einsatz sie bisher geleistet haben, wie sie die Arbeit in und Organisation von Open-Source-Projekten empfinden und welches ihr ursprünglicher Anlass war, sich in die-

sem Bereich zu betätigen. Im letzten Teil werden demographische Daten sowie Angaben über die Freizeitbeschäftigung der Probanden erfragt.

Das Besondere an der FASD-Studie ist, dass gleichzeitig Open-Source-Programmierer wie auch Entwickler an kommerziellen Projekten befragt werden. Diese Versuchsanordnung gestattet es in einem ersten Schritt zu verifizieren, ob die Programmierer tatsächlich mehr Spaß haben, mehr Flow erleben, wenn sie für Open Source entwickeln, als wenn sie diese Tätigkeit unter kommerziellen Bedingungen ausüben. Weiter ist es durch statistische Methoden möglich, das Modell zu überprüfen und auf diese Weise den Beitrag von Spaß am Engagement zu quantifizieren. Konkret werden die Angaben über das Flow-Erleben mit den Daten bezüglich Einsatzbereitschaft gemäß dem Modell korreliert, und daraus wird berechnet, welchen Anteil des Engagements durch das Modell erklärt werden kann. In einem letzten Schritt soll mit der Untersuchung gezeigt werden, dass die identifizierten Unterschiede bezüglich Projektmanagement, formaler Autorität, Anreizsystem und Abgabeterminen tatsächlich die ausschlaggebenden Faktoren sind für unterschiedliches Flow-Empfinden.

4. Weiterführende Fragen

Auf Grund der vorliegenden Forschungsresultate und der verschiedenen laufenden und geplanten Studien im Bereich Open Source bin ich der Ansicht, dass in absehbarer Zeit genügend Informationen vorhanden sein dürften, um die Motivation von Open-Source-Entwicklern hinreichend zu verstehen. Damit dürfte die Frage nach den Motiven an Interesse verlieren und sich der Forschungsschwerpunkt auf andere Phänomene im Open-Source-Bereich verlagern. Eine interessante und noch wenig erforschte Frage ist beispielsweise, unter welchen Umständen ein Open-Source-Projekt erfolgreich ist.

Das Engagement von Softwareprogrammierer ist keineswegs eine hinreichende Bedingung für einen Projekterfolg. Die empirischen Studien von Krishnamurthy (2002) und Healy und Schussman (2003) belegen, dass ein erheblicher Anteil von Open-Source-Projekten kaum das Anfangsstadium überwindet. Der Großteil der Projekte hat auch nach erheblicher Zeit noch keine Entwicklergemeinschaft anziehen können, statt dessen dämmern die Projekte als Ein-Personen-Vorhaben dahin und sind weit von einem nützlichen Einsatz entfernt.

Die Fragen nach den Erfolgsbedingungen von Open-Source-Projekten hängt möglicherweise eng mit der Frage zusammen, mit welchen Geschäftsmodellen unter welchen Bedingungen mit Open-Source-Software Geld verdient werden kann. Hecker (2000) hat eine Reihe von Geschäftsmodellen beschrieben, die nachvollziehbar aufzeigen, dass auch mit Software, deren Quellcode freigegeben ist, ein Profit erzielt werden kann. Allerdings fehlen noch weitgehend vertiefte Untersuchungen, ob die identifizierten Geschäftsmodelle auch wirklich funktionieren (siehe etwa Wichmann 2002). Die Untersuchung von Bonaccorsi und Rossi (2003) zeigt aber deutlich, dass sich die Motive, die eine Firma veranlassen, sich im Open-Source-Bereich zu engagieren, klar von denjenigen der einzelnen Hacker unterscheiden.

Sollte sich nun, z.B. mit Hilfe der FASD-Studie die Hypothese erhärten, dass Open-Source-Projekte im Wesentlichen durch Spass motiviert sind, so könnten sich die Geschäftsmodelle als das Verbindungsglied erweisen, welche den Erfolg von Open-Source-Projekten erklären. Softwareentwickler, die Spaß suchen und als Resultat dieses Verlangens Programme herstellen, befriedigen in erster Linie ihre eigenen Bedürfnisse. Erfolgreiche Software, ob unter einer Open-Source-Lizenz oder proprietär, zeichnet sich aber dadurch aus, dass sie die Bedürfnisse einer möglichst großen Benutzerschaft erfüllt. Erst der Wunsch, mit der Software auch etwas zu verdienen, zwingt die Softwareentwickler, auch noch andere als die eigenen Bedürfnisse wahrzunehmen. Und erst eine zufriedene Kundschaft führt zu einer wahrnehmbaren Marktdurchdringung des Open-Source-Produkts und damit zu dessen Erfolg.

5. Zusammenfassung

In diesem Beitrag habe ich gezeigt, dass sich das Paradox „Warum gibt eine Person ihre Software frei, wenn mit dieser Software viel Geld verdient werden könnte?“ auflöst, wenn als Aktoren nicht Profit maximierende Softwarefirmen, sondern die einzelnen Open-Source-Hacker untersucht werden. Solche Softwareentwickler sehen sich einer Low-Cost-Situation gegenüber. In dieser Situation ist es mit geringen Anreizen möglich, die Aktoren zu einem Beitrag für ein öffentliches Gut zu bewegen.

Ein Softwareentwickler kann aus einem ganzen Bündel an Motivationen auswählen, die auf unterschiedliche Weise dahin wirken, dass es für einen Programmierer vorteilhaft ist, sich in einem Open-Source-Projekt zu engagieren. In diesem Beitrag habe ich als Motive Gebrauch, Signalproduktion, Gruppenidentifikation, Lernen, Spenden und Spaß vorgestellt. Die vorliegenden empirischen Studien über das Open-Source-Phänomen bestätigen die Existenz dieser Motive. Es besteht allerdings noch keine Klarheit, wie relevant die einzelnen Motivationen zur Erklärung des Engagements der Open-Source-Hacker sind.

Diese Lücke soll mit dem Projekt FASD, zumindest teilweise, geschlossen werden. Ziel dieses Forschungsprojekts ist es, die Bedeutung der Motivation „Spaß“ quantitativ zu bestimmen. Spaß wird im FASD-Projekt mit Hilfe des Flow-Konzepts von Csikszentmihalyi untersucht. Mit Hilfe eines Online-Fragebogens soll verifiziert werden, dass Programmieren in einem Open-Source-Projekt mehr Spaß macht als unter den Bedingungen in einer kommerziellen Firma, und dass die für Open-Source-Projekte wesentlichen Eigenheiten bestimmend sind für diesen Unterschied. Weiter soll mit diesen Daten untersucht werden, welcher Anteil des Engagements mit dem auf Spaß basierenden Modell erklärt werden kann.

Kann durch die FASD-Studie gezeigt werden, dass Spaß ein zentraler Motivationsfaktor ist, so kann Open-Source-Software interpretiert werden als mehr oder weniger intendiertes Nebenprodukt einer Tätigkeit, die eminent Spaß macht. Damit stellt sich aber die Frage, warum und unter welchen Umständen Open-Source-Software erfolgreich ist. Zur Klärung dieser Frage müssen die möglichen Geschäftsmodelle für Open-Source-Software untersucht werden. Damit geraten die Firmen, die

Open-Source-Software in irgendeiner Form unterstützen, wieder ins Blickfeld der wissenschaftlichen Untersuchung.

Während also zur Erforschung der Frage, warum Open-Source-Software entsteht, die Benutzer-Programmierer untersucht werden, sind es bei der Frage, warum Open-Source-Projekte erfolgreich sind, die Firmen und ihre Geschäftsmodelle.

Anhang

Tabelle 1: Empirische Untersuchungen zu Open Source

Studie und Thema	Methode	Sample-Größe
Robles u. a. (2001); Merkmale von Open-Source-Entwicklern	Online-Fragebogen	5478
Ergebnisse: 80% der OS-Entwickler sind im IT-Bereich tätig, 33% im universitären Bereich. 21% der OS-Entwickler verdienen Geld mit dieser Tätigkeit. 54% der OS-Entwickler haben von Engagement im Open Source-Bereich beruflich nicht profitiert, wobei 26% hoffen, dies werde in Zukunft geschehen. 46% habe profitiert, wovon 15% ihre aktuelle Position diesem Engagement verdanken und 2% eine Lohnsteigerung.		
Jorgensen (2001); Wirkung des Open-Source-Entwicklungsmodells auf Motivation	Online-Fragebogen und Einzelinterviews	72
Ergebnisse: 43% der Befragten wurden für ihre Arbeit am FreeBSD-Projekt ganz oder teilweise bezahlt. 86% der Befragten geben an, dass ihre Beiträge einen Review erhalten haben. Gemäß 57% der Befragten haben die Reviews der eigenen Beiträge dazu geführt, dass ihre Fähigkeiten als Entwickler deutlich verbessert wurden.		
Hars und Ou (2001); Motivation von Open-Source-Entwicklern	Online-Fragebogen	81
Ergebnisse: Als Motivation bewerteten 80% der Befragten Selbstbestimmung als hoch bis sehr hoch, für 88% war die Bildung von Humankapital wichtig. Selbstbestimmung ist für 93% der OS-Entwickler aus dem IT-Bereich, die in ihrer Freizeit OS entwickeln, überdurchschnittlich motivierend, für bezahlte OS-Programmierer mit 61,5% nur unterdurchschnittlich. Der Lerneffekt ist für Studierende und Hobbyprogrammierer überdurchschnittlich wichtig (97%).		
Dempsey u. a. (2002); Mermale von Linux-Entwicklern	Analyse von Linux Software Maps (LSM)	4.633
Ergebnisse: Anzahl Beiträge pro Entwickler: Von den insgesamt 2.429 registrierten Entwicklern leisteten 91% 1 bis 2 Beiträge, 2,2% mehr als 5 Beiträge. Nur von 13 Entwickler stammten mehr als 10 Beiträge.		

Studie und Thema	Methode	Sample-Größe
Ghosh u. a. (2002); Merkmale von Open-Source-Entwicklern	Online-Fragebogen	2.784
Ergebnisse: Alter der OS-Entwickler: Mittelwert: 27 Jahre, 25% älter als 30 Jahre. Nationalität: 71% Europa (16,5 Frankreich, 12,4 Deutschland), 13% Nordamerika (10,3% USA). Zeitliches Engagement pro Woche: 23% < 2 h, 70% < 10 h, 14 % > 10 h und < 20 h, 9% > 20 h und < 40 h, 7% > 40 h. Motive für Engagement in OS: 80% neue Fähigkeiten erwerben, 50% Wissen teilen, 33% Kooperation in OS, 33% Verbesserung von bestehender OSS, 30% OSS als öffentliches Gut.		
Krishnamurthy (2002); Merkmale von Open-Source-Projekten	Analyse der 100 aktivsten SourceForge-Projekte	100
Ergebnisse: Die 100 erfolgreichsten SourceForge-Projekte haben im Durchschnitt eine Gruppengröße von 6,6 Entwicklern (Median: 1 Entwickler). Je mehr Entwickler ein Projekt hat, desto mehr wird es beachtet (Korrelationskoeff. 0,56) und desto häufiger erfolgen Downloads (0,27).		
Bonaccorsi und Rossi (2003); Merkmale von Firmen im Open-Source-Bereich	Fragebogen	146
Ergebnisse: Die befragten Firmen bewerteten auf einer fünfpunktigen Likertskala (Wertung von 1 bis 5) die ökonomischen Motive am höchsten (3,56), vor den technologischen (3,51) und den sozialen (3,39).		
Hertel u. a. (2003); Motivation unter Linux-Kernel-Entwicklern	Online-Fragebogen	141
Ergebnisse: Verschiedene Motivationsfaktoren (Identifikation mit dem Projekt, Spaß, Problemlösung etc.) wurden mit einer fünfpunktigen Likertskala (Wertung von 1 bis 5) bewertet und die Antworten mit dem Engagement für das Projekt korreliert. Spaß wurde mit 4,6 am höchsten bewertet. In der Varianzanalyse erhielt das Motiv „Identifikation mit der Projektgruppe“ mit 22,9% den höchsten Erklärungsgehalt. Das Ergebnis ist statistisch signifikant.		
Healy und Schussman (2003); Merkmale von Open-Source-Projekten	Analyse von SourceForge-Projekten	46.356
Ergebnisse: Projektgröße: Mittelwert: 1,7 Entwickler, Median: 1 Entwickler, 95-Perz.: 5 Entwickler. Anzahl Downloads: Mittelwert: 2289, Median: 0, 90-Perz.: 872.		
Lakhani und Wolf (2003); Motivation von Open-Source-Entwicklern	Online-Fragebogen	648

Studie und Thema	Methode	Sample-Größe
Ergebnisse: Die befragten Entwickler verwenden im Durchschnitt 14 Stunden pro Woche für die Entwicklung von OSS. 61% der Befragten erlebten bei ihrer OS-Tätigkeit die kreativsten Momente in ihrem Leben. 73% erlebten häufig oder immer Flow-Zustände bei ihrer OS-Tätigkeit. Motivation für das Engagement: Gebrauch 59%, Spaß 45%, Lernen 41%, Altruismus 33%. Die verschiedenen Motivationstypen wurden mit dem zeitlichen Engagement korreliert und der Effekt der einzelnen Beiträge wurde mittels Regressionsanalyse berechnet: Kreativität: 1,6, Bezahlung: 0,9, Freude an Teamarbeit: 0,8, Reputation: 0,6, Arbeit in anderen FOSS-Projekten: -1,6, IT-Ausbildung: -0,6 ($r^2 = 0,18$)		

Literaturverzeichnis

Backes-Gellner, U. / Lazear, E. P. / Wolff, B. (2001): *Personalökonomik. Fortgeschrittene Anwendungen für das Management*, Stuttgart

Bonaccorsi, A., C. Rossi (2003): *Altruistic individuals, selfish firms? The structure of motivation in Open Source software*,
online
http://opensource.mit.edu/papers/bnaccorsirossimotivationshort.pdf

Brooks, Frederick P. (1995): *The Mythical Man-Month: Essays on Software Engineering*, Reading, Massachusetts

Csikszentmihalyi, M. (1975): *Beyond boredom and anxiety*, San Francisco

Csikszentmihalyi, M. (1990): *Flow. The Psychology of Optimal Experience*, New York

Dempsey, B. / Weiss, D. / Jones, P. / Greenberg, J. (2002): *Who is an Open Source Software Developer?*, Communications of the ACM 45:2, p. 67–73.

Franck, E. / Jungwirth, C. (2001): *Open versus Closed Source. Eine organisationsökonomische Betrachtung zum Wettbewerb der Betriebssysteme Windows und Linux*;
online http://www.unizh.ch/ifbf/webFuehrung/Dokumente/Working Paper/FranckJungwirthOpenversusClosedSourceWP4.pdf

Franck, E., / Jungwirth, C. (2002a): *Reconciling investors and donators. The governance structure of open source*,
online http://www.unizh.ch/ifbf/webFuehrung/Dokumente/Working Paper/FranckJungwirthInvestorsAndDonatorsWP8.pdf

Franck, E. / Jungwirth, C. (2002b): *Die Governance von Open-Source-Projekten*,
online http://www.unizh.ch/ifbf/webFuehrung/Dokumente/Working Paper/FranckJungwirthGovernanceofOpenSourceWP9.pdf

Franke, N. / von Hippel, E. (2002): *Satisfying Heterogenous User Needs via Innovation Toolkits: The Case of Apache Security Software*,
online http://opensource.mit.edu/papers/frankevonhippel

Ghosh, R.A. / Glott, R. / Krieger, B. / Robles, G. (2002): *FLOSS: Free/Libre and Open Source Software: Survey and Study*,
online http://www.infonomics.nl/FLOSS/report/

Hansmann, H. B. (1980): *The role of nonprofit enterprise*, Yale Law Journal 89, 835–901

Hars, A. / Ou, S. (2001): *Working for Free? – Motivations of Participating in Open Source Projects*, 34th Annual Hawaii International Conference on System Sciences
Healy, K. / Schussman, A. (2003): *The Ecology of Open Source Software Development*, online http://opensource.mit.edu/papers/healyschussman.pdf
Hecker, F. (2000): *Setting Up Shop: The Business of Open-Source Software*, online http://www.hecker.org/writings/setting-up-shop.html
Hertel, G. / Niedner, S. / Herrmann, S. (2003): *Motivation of Software Developers in Open Source Projects*, Research Policy 32:7, S. 1159–1177
Jorgensen, N. (2001): *Putting it All in the Trunk*, Information Systems Journal 11:4
Krishnamurthy, S (2002): *Cave or Community? An Empirical Examination of 100 Mature Open Source Projects*, online http://www.firstmonday.org/issues/issue7_6/krishnamurthy/index.html
Lakhani, K. / Wolf, R. (2003): *Why Hackers Do What They Do: Understanding Motivation Effort in Free/Open Source Software Projects*, online http://opensource.mit.edu/papers/lakhaniwolf.pdf
Lerner, J. / Tirole, J. (2001): *The Simple Economics of Open Source*, online http://opensource.mit.edu/papers/
Lorenz, W. / Wagner, J. (1988): *Kompensierende Lohndifferentiale*, WiSt-Wirtschaftswissenschaftliches Studium 17:10, S. 515–518
Osterloh, M. / Rota, S. / Kuster, B. (2002): *Open Source Software Production: Climbing on the Shoulders of Giants*, online http://opensource.mit.edu/papers/osterlohrotakuster.pdf
Raymond, E. S. (2000): *Homesteading the Noosphere*, online http://www.catb.org/~esr/writings/homesteading/
Remy, K. (2002): *Entwicklung eines Fragebogens zum Flow-Erleben*, Bielefeld: Diplomarbeit (Fakultät für Psychologie und Sportwissenschaft)
Rheinberg, F. (1997): *Motivation*, Stuttgart
Robles, G. / Scheider, H. / Tretkowski, I. / Weber, N. (2001): *Who is doing it? A research on Libre Software developers*, online http://widi.berlios.de/paper/study.html
Toffler, A. (1981): *The Third Wave*, New York
Torvalds, L. / Diamond, D. (2001): *Just for FUN. The Story of an Accidental Revolutionary*, New York
Hippel, E. von / Krogh, G. von (2002): *Exploring the Open Source Software Phenomenon: Issues for Orgnization Science*, online http://opensource.mit.edu/papers/removehippelkrogh.pdf
Weber, Steven (2000): *The Political Economy of Open Source Software*, online http://brie.berkeley.edu/~briewww/pubs/wp/wp140.pdf
Wichmann, T. (2002): *FLOSS Final Report – Part 2: Firms' Open Source Strategy: Motivations and Policy Implications*, online http://www.berlecon.de/studien/downloads/200207FLOSS_Activities.pdf

Stärken und Schwächen freier und Open-Source-Software im Unternehmen

THOMAS WIELAND

Zusammenfassung

Freie und Open-Source-Software stößt bei einer wachsenden Zahl von Unternehmen auf Interesse, wobei die meisten lediglich an eine Nutzung bestehender Software denken, manche aber auch eigene Entwicklungen freigeben wollen. Damit die Entscheidung für den Einsatz von bzw. das Engagement für freie und Open-Source-Software nicht nur „aus dem Bauch heraus" getroffen werden muss, ist eine sorgfältige Analyse der individuellen Situation der Firma notwendig. Diese setzt natürlich voraus, dass nicht nur die Besonderheiten dieser Art von Software bekannt sind, sondern auch deren Stärken und Schwächen, Potenziale und Bedrohungen. In diesem Beitrag soll ein knapper Überblick über die wichtigsten Aspekte aus diesen vier Kategorien gegeben werden, um den unternehmerischen Entscheidungsprozess zu erleichtern und einige Hilfestellungen zur Auswahl zu geben. Dabei stehen naturgemäß hauptsächlich strategische und wirtschaftliche Gesichtspunkte im Vordergrund; die unbestreitbaren technischen Vorteile werden an anderer Stelle in diesem Band bereits ausführlich gewürdigt.

Einleitung

Bei immer mehr Firmen und Behörden wächst das Interesse an freier und Open-Source-Software (im Folgenden kurz OSS). Zunächst nutzen sie sie lediglich, wie sie bisher auch proprietäre Software genutzt haben, beispielsweise aus Gründen der Kostensenkung. Gerade Entwickler in Industrie- oder Dienstleistungsunternehmen stellen oft jedoch mit der Zeit fest, dass die eine oder andere OSS-Bibliothek oder -Anwendung eine ausgezeichnete Grundlage für die eigene Produktentwicklung darstellen würde. Damit wird die Verflechtung mit der so genannten „Open-Source-Szene" immer enger und natürlich auch komplizierter. Schließlich kommt auch eine wachsende Zahl von Unternehmen zu dem Schluss, dass sie eigene Software als Open Source freigeben wollen.

Die Diskussion um Open Source wird mittlerweile in einer so breiten Öffentlichkeit geführt, dass so ziemlich jeder Verantwortliche in Unternehmen, die Informationstechnologien einsetzen, schon einmal mit dem Thema in Kontakt kam. Die Gefühlslage kann dabei schnell von einer Ablehnung aus einer vagen Unsicherheit heraus bis hin zu Begeisterung für die Idee schwanken. Der derzeitige allgegenwärtige Hype für OSS trägt noch ein Übriges dazu bei. Doch unternehmerische Entscheidungen sollten nicht aus unbestimmbaren Gefühlen heraus getroffen werden – und auch nicht nur als Nachahmung eines Trends. Jeder Verantwortliche sollte an-

hand einer Reihe von konkreten Argumenten für seinen Einzelfall entscheiden, ob der Einsatz von bzw. das Engagement für freie und Open-Source-Software in seiner Firma sinnvoll ist oder nicht.

Eine klassische Technik, die für die Untersuchung solcher Entscheidungssituationen gerne verwendet wird, ist die so genannte SWOT-Analyse. Diese vier Buchstaben stehen dabei für die englischen Begriffe strengths, weaknesses, opportunities und threats – also Stärken, Schwächen, Potenziale und Bedrohungen. Nach diesem Schema wollen auch wir hier untersuchen, welche Argumente für und gegen freie und Open-Source-Software im Unternehmen sprechen. Dabei soll sowohl der reine Nutzungsfall als auch der des Mitwirkenden an OSS betrachtet werden.

In diesem Beitrag wollen wir eine Software als „Open-Source-Software" bezeichnen, wenn die Lizenz für ihre Verbreitung und Nutzung mit der Open-Source-Definition in Einklang steht, die von der Open-Source-Initiative (2003) veröffentlich wurde. Diese Definition umfasst zehn klare und strenge Regeln, die eine Software und ihre Lizenz erfüllen müssen, damit die Bezeichnung „Open Source" gerechtfertigt ist. Bereits die Lizenzmodelle, die von dieser Definition umfasst werden, sind äußerst vielfältig. Ein noch viel breiteres Spektrum machen aber die Projekte selbst aus. Allein beim Mediator SourceForge[1], einem Portal für OSS-Projekte, waren Ende 2003 über 70.000 verschiedenen Projekte registriert. Diese unterschieden sich sehr stark in Größe, d. h. Zahl der Mitwirkenden bzw. Umfang der entwickelten Software, und Status, von vagen Ideen über viel versprechende Anfänge bis zu reifen Projekten.

Daraus ist ersichtlich, dass alle Aussagen, die im Folgenden zu treffen sind, immer nur für einen Teil von Projekten Gültigkeit besitzen. Für jedes Argument für oder gegen OSS existieren sicherlich mehrere Gegenbeispiele, die genau diesen Gesichtspunkt widerlegen. Es muss sich daher also nicht um eine Damm öffnende Grundsatzentscheidung handeln, wenn ein Unternehmen sich zum Einsatz einer bestimmten Open-Source-Software entschließt. Es muss nach wie vor in jedem Einzelfall abgewogen werden, welcher Schritt aus unternehmerischer Sicht sinnvoller ist.

Genauso wenig, wie es „das Open-Source-Projekt" gibt, gibt es „die Open-Source-Community". Eine der verbreiteten grundlegenden Fehlannahmen ist nämlich, die Community als eine einzige einheitliche Menge von Menschen zu betrachten. Zu jedem Projekt bildet sich vielmehr eine eigene Community heraus. Diese kann sich mit der anderer Projekte überlappen, wenn sich Entwickler etwa in mehreren Projekten engagieren, ist aber mit den meisten völlig disjunkt. Sie kann nur aus einer Person bestehen oder aber aus vielen Tausenden.

Für unsere Zwecke hier definieren wir also:

- Eine „Projekt-Community" ist ein virtuelles Team, das sich zeitweise auf freiwilliger Basis zusammensetzt, um eine bestimmte Aufgabe gemeinsam zu lösen.
- Die „Community" ist die Gesamtheit aller Projekt-Communities.

[1] SourceForge: http://sourceforge.net.

Zu beachten ist, dass die Grenzen hier fließend sind, abhängig von den Rollen, die ein Individuum in Bezug auf das jeweilige Projekt spielt. Ist eine Person, die die Software unverändert nutzt, bereits Mitglied der Community? Nach obiger Definition nicht. Aber ist jemand, der die Software an seine Bedürfnisse anpasst, ein Mitglied? Schon eher, wenn man den Begriff der „Aufgabe" weit genug fasst.

Für das Weitere wollen wir den Begriff der Community in der Tat recht weit fassen. Es geht darum, nicht nur alle Entwickler zu erfassen und zu erreichen, die bereits einen Beitrag leisten, sondern das Forum hervorzuheben, in dem Nutzer und Entwickler sowie alle, die potenzielle Beiträge zur Verfügung haben, sich austauschen, Probleme diskutieren, Anregungen und Vorschläge machen sowie Lob und Kritik aussprechen können. Eben in dieser Art von Forum unterscheidet sich OSS in der Tat grundsätzlich von klassischer proprietärer Software.

Stärken freier Software

Open-Source-Software würde nicht auf so viel Interesse stoßen, wenn sie nicht einige unbestreitbare Stärken vorzuweisen hätte. Diese wollen wir zunächst betrachten.

Einer der renommiertesten Vorreiter der Open-Source-Bewegung ist Eric Raymond, der mit seinen Essays „The Cathedral and the Bazaar" und „The Magic Cauldron" (beide abgedruckt in Raymond 1999) 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 Codes 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 Infrastrukturkomponenten wie Betriebssystemkernels, Treiber oder Netzwerkdienste sinnvoll.

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 unabhängigen Seiten als sicher eingestuft werden, kann letztendlich vertraut werden. Aus diesem Grund setzen Organisationen, bei denen Sicherheit eine große Rolle spielt, wie der amerikanische Geheimdienst NSA, das deutsche Bundesamt für Sicherheit in der Informationstechnik sowie das Auswärtige Amt immer häufiger von ihnen selbst als sicher zertifizierte OSS-Systeme ein.

Zudem folgt die Gesamtkonzeption bei freier Software meist einem strengeren Sicherheitsmodell. Insbesondere das Betriebssystem GNU/Linux mit seinen verschiedenen Infrastrukturkomponenten auf Open-Source-Basis erlaubt eine scharfe Trennung zwischen Rechten des Anwenders und des Administrators. Hierzu kommt beispielsweise das Bundesinnenministerium in einer Studie (Stör, 2003) zu dem Schluss: „Beim Systemmanagement folgen die OSS-Betriebssyteme ihrer Herkunft entsprechend dem UNIX-Weg. […] Für Administratoren und Betriebsorgani-

sation sind bei einer Migration auch konzeptuelle Änderungen zu erwarten, die insbesondere bezüglich der Sicherheit große Fortschritte ermöglichen. Die Sicherheit und Zuverlässigkeit, die den Linux-Systemen allgemein zugeschrieben wird, ist nicht zuletzt das Resultat des Systemmanagement."

Die Offenheit ist aber auch in anderer Hinsicht eine Stärke. 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.

Darüber hinaus hat sich gezeigt, dass einmal entdeckte Fehler und Sicherheitsprobleme bei OSS im Durchschnitt etwa sechsmal schneller behoben werden als bei gleichartiger proprietärer Software (LW 2003). So zeigte etwa eWeek (2002) anhand einer Reihe von Beispielen, dass Sicherheitsprobleme in OSS-Projekten sehr ernst genommen und äußerst schnell behoben werden, während Hersteller kommerzieller Software sich oft zunächst um eine Verharmlosung des Problems bemühen.

Auch ist es bei OSS normalerweise nicht nötig, die gesamte Anwendung zu aktualisieren, nur um punktuelle Fehlfunktionen zu beheben. Der Benutzer kann also bei Problemen stets auf eine rasche und zielgerichtete Hilfe hoffen. Bei größeren und sehr aktiven Projekten reagieren die Entwickler im Allgemeinen auf E-Mail-Anfragen aller Art innerhalb weniger Stunden mit einer konstruktiven Antwort.

Schwächen freier Software

Natürlich ist aber auch freie und Open-Source-Software nicht immer ohne Probleme. 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 – obschon selbst das zuweilen der Fall ist. 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.

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, darunter natürlich auch die großen Linux-Distributoren Red Hat 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 – theoretisch – jederzeit austauschen.

Auch die Softwarepakete selbst haben manchmal noch nicht den Grad der Reife erreicht, damit sie für den Alltagseinsatz im Unternehmen geeignet sind. Die Schwierigkeit für den potenziellen Nutzer besteht hier bereits darin, den Reifegrad überhaupt zu erkennen. Versionsnummern allein sagen dazu kaum etwas aus; eine Version 0.4 einer Software kann ein stabileres Programm bedeuten als eine 1.1 einer anderen. Zum Glück setzen sich auch in Open-Source-Projekten immer mehr etablierte Techniken des „software engineering" durch, etwa die des „release management". Wird ein Entwicklungsstand beispielsweise als „stable release" bezeichnet, kann man im Allgemeinen von einer stabilen Version ausgehen. „Unstable" oder „snapshots" ist dagegen eher mit Vorsicht zu begegnen. Der hohe Anspruch vieler OSS-Entwickler führt jedoch ab und an dazu, dass selbst ausgereifte Programme noch als „unstable" klassifiziert werden, die bei einem kommerziellen Entwicklungszyklus schon lange auf dem Markt wären.

Der erste Schritt bei der Auswahl eines OSS-Produkts für den Firmeneinsatz ist also die umfassende Evaluation, die gegebenenfalls auch ein Dienstleister durchführen kann. In dieser sollte durch eingehende Tests festgestellt werden, ob die Software in der aktuellen Version die in sie gesetzten Erwartungen auch erfüllen kann.

Natürlich muss man sich vorher schon sicher sein, das richtige Programm ausgewählt zu haben. Denn auch das ist eine „Schwäche" von OSS – wenn man so will. Für fast alle Aufgaben gibt es mehr als ein Projekt, das eine Lösung dazu erarbeiten will. Der Ansatz kann dabei ebenso unterschiedlich sein wie der Umfang und der Reifegrad. Auf den ersten Blick kann man daher kaum erkennen, welche Software nun besser geeignet ist. Kriterien für die Auswahl können u.a. sein:

- Zugesicherte oder getestete Funktionalität
- Stabilität
- Qualität von Architektur und Design
- Lizenzmodell
- Unterstützung durch die Projekt-Community
- Zahl der Mitarbeiter und Dauer des Projekts
- Engagement von Firmen.

Ein Fallbeispiel für die Auswahl eines OSS-Pakets wurde etwa von Hang et al. (2004) beschrieben.

Ein weiteres immer wieder auftauchendes Problem bei freier Software ist die gegenseitige Abhängigkeit. Viele Entwickler versuchen, sich mit den Programmierern anderer OSS-Projekte zu koordinieren, um so aufeinander aufbauen zu können. Das führt für den Anwender dazu, dass die Version, die er ausgewählt hat oder die er auch nur ausprobieren möchte, auf seinem Zielsystem nicht läuft, da sie von einer anderen Software abhängt, die augenblicklich nicht installiert ist. Erschwerend kommt leider häufig dazu, dass die nun für die Installation benötigten Programme zwar vorhanden sind, aber in einer älteren Version. Um ein Update für diese einzuspielen, brauchen sie wiederum aber Updates von weiterer Software, von der sie selbst abhängen. Das kann sich über mehrere Stufen fortsetzen, bis der Anwender feststellt, dass er ein vollständiges Upgrade seiner Betriebssystemplattform bräuchte, um ein bestimmtes OSS-Paket überhaupt starten zu können.

Hierbei hilft nur, sich frühzeitig darüber klar zu werden, welche Programme in welchen Versionen geschäftskritisch sind und wie weit die Bereitschaft zum Update geht. Sind die Erfordernisse, die eine bestimmte Software stellt, zu umfangreich, muss auf die Evaluation beziehungsweise Nutzung dieser Software verzichtet werden – sofern nicht in Absprache mit den Entwicklern eine Zwischenlösung gefunden werden kann.

Potenziale freier Software

Was bedeuten diese Eigenschaften von freier und Open-Source-Software nun für den Einsatz im Unternehmen? Welche Potenziale können damit erschlossen werden, und welche strategischen Vorteile ergeben sich daraus? Mit diesen Fragen wollen wir uns nun beschäftigen.

Einer der sicher auffälligsten Vorteile von freier Software sind die geringen Kosten. Dabei ist „frei" im Grunde im Sinne von „freier Rede" und nicht von „Freibier" zu verstehen. 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 (Free Software Foundation, 1999). Es ist nicht einmal untersagt, dass ein Anbieter Open-Source-Software gegen Entgelt verkauft. Nur darf er keinem anderen verbieten, die gleiche Software kostenlos abzugeben. Daher hat es sich heute eingebürgert, dass nur für die Bereitstellung, die Aufarbeitung und den Datenträger Gebühren verlangt werden. Nichtsdestotrotz kann man alle freie und Open-Source-Software kostenfrei als Download über das Internet erhalten – von den Gebühren für den Internetzugang einmal abgesehen.

In der Praxis kann man also die Anschaffungskosten für OSS-Programme vernachlässigen. Dabei spielt es nicht einmal eine Rolle, ob nun eine Kopie des Programms (etwa auf einem Server) oder Dutzende bis Tausende (auf Arbeitsplatzrechnern) eingesetzt werden sollen. Dies stellt einen fundamentalen Unterschied zu kommerzieller Software dar, bei der ja pro genutzter Lizenz Gebühren anfallen.

Jeder erfahrene IT-Verantwortliche weiß indessen, dass die Lizenzkosten einer Software nur einen geringen Teil der Gesamtkosten ausmachen, die mit dieser Software verbunden sind. Dazu gehören auch noch die Kosten für die erforderlichen Hardwareressourcen, für Systemadministration und Support, für Datenkommunikation und Netzwerk, für benutzerbezogene und technische Ausfallzeiten sowie für den Aufbau oder Einkauf des nötigen Know-How. Diese Summe der gesamten Kosten eines Investionsgutes (hier also einer Software) bezeichnet man auch als „Total Cost of Ownership" (TCO).

Es gibt eine ganze Reihe von Studien, in der die TCO von proprietärer und freier Software miteinander verglichen werden. Die meisten davon stellen GNU/Linux und Windows oder ein kommerzielles Unix gegenüber. So kommen beispielsweise Gillen et al. (2001) zu dem Schluss, dass GNU/Linux im Bereich Intranet/Internet nur 55% der TCO-Kosten pro Jahr verursacht, die ein vergleichbares System auf RISC/Unix-Basis mit sich bringen würde. Das australische Marktanalyse-Unternehmen Cybersource (2002) stellt bei einem ähnlichen Vergleich von GNU/Linux mit

Windows eine Kostenersparnis von bis zu 34,3 % über einen Dreijahreszeitraum fest. Der Anwalt Brendan Scott (2002) hat mit eher theoretischen Überlegungen gezeigt, warum freie Software langfristig niedrigere Gesamtkosten haben muss. Und sogar der CEO von Sun Microsystems, Scott McNealy, spricht im Zusammenhang mit GNU/Linux immer wieder von bis zu 90% niedrigeren IT-Kosten (zitiert nach Nowak 2003).

Was heißt das Ganze nun für die Praxis? Hier haben Binder und Lipski (2003) recht klar herausgestellt: Es gibt ein beachtliches Einsparpotenzial durch OSS – davon profitieren allerdings große Unternehmen in sehr viel stärkerem Maße als kleine, da jene oftmals schon internes Open-Source-Know-How haben und es sich nicht teuer extern einkaufen müssen. Außerdem fallen bei großen Organisationen Lizenzkosten proprietärer Software für Tausende von Arbeitsplätzen stärker ins Gewicht. Somit sind pauschale Modellrechnungen immer nur Anhaltspunkte, die für das einzelne Unternehmen lediglich begrenzte Aussagekraft haben. Jede Firma muss daher selbst analysieren, welchen Bedarf an IT-Unterstützung sie hat und welche Kosten dafür proprietäre und OSS-Lösungen mit sich bringen.

Neben reinen Kostenargumenten gibt es aber auch eine Reihe von strategischen Aspekten, die für freie und Open-Source-Software sprechen. Baut etwa 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. Er ist also gezwungen, synchron mit dem „Innovationszyklus" des Herstellers zu bleiben, auch wenn er eigentlich keinen Bedarf an Aktualisierungen bzw. Neuerungen hat. Selbst wenn nur die Unterstützung für ein eingesetztes Produkt wegfällt, ist der Anwender zu einem Update veranlasst. Ein Beispiel ist die gegenwärtige Welle der Plattformwechsel bei Arbeitsplatzsystemen als Folge der Ankündigung der Einstellung des Supports für Windows NT 4 durch Microsoft.

Besonders kann es kritisch werden, wenn der Hersteller plötzlich nicht mehr am Markt ist – weil er in 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.

Dabei haben wir bislang nur die Potenziale von OSS für reine Anwender betrachtet. Zum Abschluss dieses Abschnitts wollen wir noch untersuchen, welche Gründe aus Sicht eines Softwareherstellers dafür sprechen, ein Programm als Open Source zu veröffentlichen (siehe auch Wieland 2001).

Früher entstanden die meisten Open-Source-Projekte dadurch, dass ein Programmierer eine Idee verwirklichen wollte oder ein bestimmtes Tool benötigte, das er selbst schreiben musste, dann aber mit anderen teilen wollte. Wenn eine Firma heute ein solches Projekt ins Leben ruft, geschieht dies kaum noch aus technischen, sondern meist aus Marketingüberlegungen. Auf diese Weise kann sich das Unter-

nehmen nicht nur als offen und innovativ hervorheben, sondern auch bestimmte Benutzergruppen an sich binden oder sich in neuen Märkten positionieren.

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 jedoch 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 OSS-Programme ergänzen beziehungsweise erweitern. Das können größere, kostspielige Softwarepakete, etwa für die Entwicklung oder Administration, sein genauso wie Hardware. Auch dafür ist IBM mit der WebSphere-Produktreihe, die auf Java- und Internetstandards sowie auf Eclipse setzt, oder der Linux-Portierung auf AS/400 ein gutes Beispiel.

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.

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 auf Grund 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.

Aber nicht nur die Entwicklung kostet Geld; oft ist der laufende Unterhalt für die Pflege einer Software sogar noch teuer. Viele Firmen pflegen Softwareprodukte 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 den Pflegeaufwand damit zumindest teilweise auf andere verteilen kann. Dazu ist allerdings auch der vorbereitende Aufbau einer entsprechenden Community nötig . Ein prominentes Beispiel für diesen Schritt ist AOL/Netscape mit der Freigabe des Mozilla-Browsers – auch wenn dabei im Laufe der Zeit noch ein paar andere Einflüsse hinzukamen.

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

teres durch andere übernehmen. Falls also einer der Mitarbeiter die Firma verläßt oder eine neue Aufgabe übernimmt (oder gar alle), 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.

Bedrohungen für freie Software

Aber selbst freie und Open-Source-Software und deren Anwender sehen sich einigen Bedrohungen gegenüber. Zunächst bleibt festzuhalten, dass die Grundidee von freier Software nicht nur den freien Zugang beinhaltete. Man stellte sich vielmehr vor, dass die unentgeltliche Software, die man von anderen erhält, durch eigene Mitarbeit an diesem oder einem anderen Projekt kompensiert wird. Auf diese Weise haben alle einen Nutzen, wobei ihr eigener Aufwand vertretbar bleibt.

Dieses Prinzip ist heute offenbar durchbrochen. Es gibt wesentlich mehr Nutzer freier Software als Entwickler. Und die Nutzer sehen darin längst nicht mehr eine moralische Verpflichtung zur Gegenleistung. Solange es trotzdem genügend Entwickler gibt, ist dagegen auch nichts einzuwenden. Denn die wachsende Verbreitung von OSS allein ist schon ein Erfolg, der viele zusätzliche positive Veränderungen mit sich bringt. Zudem haben sich die Motivationen der Teilnehmer an OSS-Projekten deutlich gewandelt. Wie Hars und Ou (2001) sowie Hertel et al. (2003) gezeigt haben, spielen neben idealistischen Beweggründen auch immer öfter materielle eine Rolle. Die Free Software Foundation schätzt, dass bereits mehr als ein Drittel der Entwicklungsarbeit in allen OSS-Projekten von Unternehmen bzw. deren Mitarbeitern geleistet wird. Die im letzten Abschnitt erwähnten Beispiele machen auch die Hintergründe dieses Engagements deutlich. Nichtsdestotrotz sollte sich im Interesse des Fortbestandes und der Weiterentwicklung der Open-Source-Idee jedes Unternehmen, das freie und Open-Source-Software nutzt, über eine Anerkennung für die erhaltene Software Gedanken machen.

Dass freie und Open-Source-Software generell wieder vom Markt verschwindet, ist also keine realistische Gefahr. Dass ein einzelnes Projekt eingestellt wird, kommt dagegen durchaus häufiger vor. Gerade weniger prominente, kleinere Projekte kämpfen oft ums Überleben. Sie haben zu wenig Zulauf von Entwicklern und können daher ihre selbst gesteckten Ziele nicht einmal erreichen. In einer von uns durchgeführten Umfrage unter verschiedenen Projekten, die in Seifert und Wieland (2003) im Detail vorgestellt wird, wurde der Mangel an Entwicklungskapazitäten gleich von mehreren Befragten genannt. Immer wieder werden Projekte nicht mehr weitergeführt, weil es an Mitarbeitern mangelt. Manchmal entscheidet sich der Projektleiter, der dann ohne Hilfe ist und neben seiner eigenen Arbeitszeit noch die

Serverkosten für das Projekt-Hosting etc. zu tragen hat, auch dafür, das Projekt nur noch als kommerzielle Version weiterzuführen. Je wichtiger einem Nutzer also die OSS-Programme sind, desto mehr sollte er durch eigenes Engagement für einen Fortbestand als Open Source sorgen.

In der öffentlichen Diskussion wird zuweilen auch der „virusartige" Charakter von Open Source als Bedrohung dargestellt. Danach soll durch den Umgang mit OSS auch andere Software des Unternehmens „infiziert" werden, sodass keinerlei Software mehr kommerziell vertrieben werden kann, sobald eine unter einer Open-Source-Lizenz steht (speziell unter der GNU General Public License, GPL). Dies ist natürlich eine völlig falsche und irreführende Behauptung, wie viele Beispiele von IBM über Hewlett Packard bis zu Sun Microsystems zeigen. Der Urheber einer Software behält in jedem Fall das Recht zu entscheiden, unter welcher Lizenz seine Software verbreitet werden soll. Das schließt auch das Recht ein, ein Produkt unter eine Open-Source-Lizenz zu stellen und ein anderes unter eine kommerzielle. Durch die Anwendung einer Open-Source-Lizenz gibt man auch nicht seine Urheberrechte auf – es ist also etwas völlig Anderes als „public domain". Es werden eben nur die Verfügungsrechte des Benutzers anders als bei proprietärer Software geregelt. Gerade aus diesen Gründen beinhaltet auch die GPL die Intention, ihre Lizenz auf von der jeweiligen Software abgeleitete Werke auszudehnen. Wenn ein Programm direkt mit GPL-Software gelinkt wird (im Sinne von Linken mit Objekt-Code), wird dieses als abgeleitetes Werk angesehen und muss es im Quellcode und unter GPL-Lizenz veröffentlicht werden (Free Software Foundation, 1999). Doch in der Praxis besteht nur selten die Notwendigkeit des direkten Linkens mit GPL-Software. Die meisten gebräuchlichen OSS-Systeme, mit denen kommerzielle Software gelinkt ist, stehen unter der so genannten Library GPL (LGPL), die explizit das Linken mit „closed source" erlaubt und die Veröffentlichung des Quellcodes nur bei Änderungen an den Bibliotheken selbst verlangt. Im Übrigen müssen sogar bei GPL-Software die Quellen nur dann offen gelegt werden, wenn die Software selbst veröffentlicht wird. Eine rein firmeninterne Nutzung fällt beispielsweise nicht darunter.

Eine weitere Schwierigkeit beim Umgang mit freier und Open-Source-Software kann aus den gegensätzlichen Einstellungen, Motivationen und Erwartungen der Nutzer beziehungsweise der firmeneigenen Entwickler und den OSS-Entwicklern erwachsen. OSS-Projekte sind üblicherweise bestimmt durch Beiträge zufälliger Programmierer, häufige kleine Releases, Feature-Auswahl nach Bereitschaft und Können der Entwickler, Dezentralisierung sowie „peer review" (Raymond, 1999). Typische Geschäftspraktiken wie Terminpläne, Aufgabenverteilung durch Projektleiter, Festlegung des Funktionsumfangs und des Auslieferdatums durch den Produktmanager sind unter OSS-Entwicklern weitgehend unbekannt (oder werden ignoriert und zuweilen verschmäht). Unternehmen, die sich eine OSS-Community zu Nutze machen und deren Software produktiv einsetzen wollen, tun daher gut daran, diese Spielregeln zu beherzigen und sich mit dieser für sie ungewohnten Kultur auseinander zu setzen (siehe auch Pavlicek 2000). Open-Source-Entwickler akzeptieren es im Allgemeinen nicht, wenn jemand versucht, ihnen seine Entwicklungs- oder Managementprozesse zu oktroyieren. Das bedeutet für die andere Seite, dass

die Firma mehr oder weniger die existierende OSS-Kultur anerkennen und sich um geeignete Schnittstellen zur Community kümmern muss (Näheres bei Seifert und Wieland 2003).

Eines der größten Hemmnisse auf dem Weg zum Einsatz von freier und Open-Source-Software sind gegenwärtig Unsicherheiten bei der Rechtslage. Insbesondere Gewährleistung und Haftung sind offene Punkte, die zwar schon von Juristen mehrfach diskutiert (etwa von Siepmann 1999 sowie von Jaeger und Metzger 2002), bislang aber nicht im Einzelfall gerichtlich entschieden wurden. Man kann sich beispielsweise trefflich darüber streiten, inwieweit etwa der generelle Haftungsausschluss von GPL und anderen Lizenzen mit dem deutschen Recht vereinbar ist – wichtig ist aber zu bedenken, welche rechtlichen Möglichkeiten bei kommerzieller Software gegeben sind. Vor allem entscheidend sind dabei nicht die prinzipiellen rechtlichen Möglichkeiten, sondern ist deren gerichtliche Durchsetzbarkeit.

Für Unternehmer stellt die rechtliche Seite des Einsatz von OSS dennoch eine Unsicherheit dar. Wenn es wirklich um geschäftskritische Anwendungen geht, sollte sich die Firma entweder selbst genügend Know-How zulegen, um den Betrieb in der benötigten Qualität zu gewährleisten, oder damit einen externen Dienstleister betrauen, in dessen Vertrag dann auch Haftungsaspekte aufgenommen werden können. Zudem ist die Lage bei Open Source immer noch deutlich besser, da bei Problemen schnell eigene Verbesserungen vorgenommen oder in Auftrag gegeben werden können. Letztendlich muss man zu bedenken geben, dass auch bei sonstigen Fehlern (und Fehlentscheidungen) im Geschäftsleben das Unternehmen selbst dafür gerade stehen können muss.

Eine generelle Bedrohung von freier und Open-Source-Software ergibt sich aber auch aus möglichen Ansprüchen Dritter an der Urheberschaft. Ein aktuelles Beispiel dafür ist die gerichtliche Auseinandersetzung der Santa Cruz Operation (SCO) mit IBM. Nach Angaben von SCO enthält Linux Codebestandteile, die geistiges Eigentum von SCO sind. Auf Basis dieser Anschuldigung hatte SCO im Juli 2003 Lizenzansprüche vor allem gegen Firmen geltend machen wollen. Jeder Einsatz von Linux solle danach eine Lizenzverletzung sein; SCO hat US-amerikanische Unternehmen bereits angeschrieben und entsprechende Lizenzgebühren eingefordert. Es ist unwahrscheinlich, dass der Streit letztlich zugunsten von SCO ausgehen wird – zu viele Fakten sprechen dagegen. Er zeigt jedoch die Verletzlichkeit der freien und Open-Source-Software. Dass eine bestimmte OSS Rechte Dritter verletzt, die dann auf deren Einhaltung bestehen, ist nie ganz auszuschließen. Für den einzelnen Nutzer handelt sich dabei zwar um ein Risiko, aber doch um ein geringes.

Fazit

Dieser Beitrag kann sicher nur einen ersten Überblick über die Besonderheiten von freier und Open-Source-Software bieten. Die Betrachtung kann daher keinen Anspruch auf Vollständigkeit erheben. Eine Fülle weiteren Materials findet sich in der angegebenen Literatur, dort besonders bei Wheeler (2003).

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 überprüfen, 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. 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. hat gezeigt, dass es am Durchbruch keine Zweifel mehr geben kann, ja dass dieser gerade schon vor sich geht.

Die gegenwärtigen Ressentiments gehen mehr auf Ängste und Befürchtungen denn auf konkrete Bedrohungen zurück. Jedes Unternehmen muss indessen im Einzelfall entscheiden, ob es für die eine oder andere IT-Aufgabe ein Softwareprodukt kaufen oder eine freie Alternative einsetzen will. Insbesondere der Kosten-Nutzen-Vergleich muss individuell vorgenommen werden und kann nicht durch Modellrechnungen aller Art ersetzt werden. Die freien und Open-Source-Programme sollten allerdings bei jeder Neuanschaffung oder -ausrichtung in Betracht gezogen werden – und das nicht allein aus Kostengründen.

Danksagungen

Der Autor dankt den Kooperationspartnern bei Siemens AG, Siemens Business Services, 4Soft und der Technischen Universität München für die gute und fruchtbare Zusammenarbeit und die anspornenden Diskussionen.

Literatur

Binder, S., C. Lipski (2003): *Kassensturz: Open-Source und proprietäre Software im Vergleich*, Soreon Research

Cybersource (2002): *Linux vs. Windows – Total Cost of Ownership Comparison*, online http://www.cyber.com.au/cyber/about/linux_vs_windows_pricing_comparison.pdf, zuletzt am 20.11.2003 zugegriffen

Free Software Foundation (1999): *Licenses of free software*, online http://www.gnu.org/licenses/, zuletzt am 20.11.2003 zugegriffen

Gillen, A. / Kusnetzky, D / McLarnon, S. (2001): *The Role of Linux in Reducing the Cost of Enterprise Computing*, IDC White Paper, Framingham

Hang, J. / Hohensohn, H. / Mayr, K. / Wieland, T. (2004): *Benefits and Pitfalls of Open Source in Commercial Contexts*, in: Koch, S. (Hrsg.): Free/Open Source Software Development. Hershey, PA

Hars, A., S. Ou (2001): *Working for free? – Motivations for participating in open source projects*, Proceedings 34th HICSS Conference, Maui

Hertel, G., S. Niedner, S. Herrmann (2003): *Motivation of software developers in open source projects: An internet-based survey of contributors to the Linux kernel*, Research Policy, Special Issue

Jaeger, T. / Metzger A. (2002): *Open Source Software – Rechtliche Rahmenbedingungen der Freien Software*, München

Nowak, P. (LW 2003): *The Powerful Economic Underpinnings of OSS*, LinuxWorld 2003, 17.10,
online http://www.linuxworld.com/story/34293.htm, zuletzt am 20.11.2003 zugegriffen

Open Source Initiative (2003): *The Open Source Definition (Version 1.9)*,
online http://www.opensource.org/docs/definition.php, zuletzt am 19.11.2003 zugegriffen

Pavlicek, R. (2000): *Embracing Insanity: Open Source Software Development*, Indianapolis

Rapoza, J. (2002): *eWeek Labs: Open Source Quicker at Fixing Flaws*, eWeek, 30.9.2002
online http://www.eweek.com/article2/0,3959,562226,00.asp, zuletzt am 21.11.2003 zugegriffen

Raymond, Eric S. (1999): *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, Sebastopol

Scott, B. (2002): *Why Free Software's Long Run TCO must be lower*,
online http://www.members.optushome.com.au/brendanscott/papers/freesoftwaretco150702.html, zuletzt am 22.11.2003 zugegriffen

Seifert, T. / Wieland, T. (2003): *Prerequisites For Enterprises To Get Involved In Open Source Software Development*, in: 1st Workshop of Open Source Software in Industrial Environments, Proceedings of Net.ObjectDays 2003, Erfurt

Siepmann, J. (1999): *Lizenz- und haftungsrechtliche Fragen bei der kommerziellen Nutzung Freier Software*, LinuxTag 1999,
online http://www.kanzlei-siepmann.de/linux-tag/vortrag.html, zuletzt am 23.11.2003 zugegriffen

Stör, Michael (2003): *Der Migrationsleitfaden*, Bonn

Wheeler, D.A. (2003): *Why Open Source Software / Free Software (OSS/FS)? Look at the Numbers!*,
online http://www.dwheeler.com/oss_fs_why.html, zuletzt am 23.11.2003 zugegriffen

Wieland, T. (2001): *Open Source im Unternehmen*, in: A. v. Raison, R. Schönfeldt (Hrsg.): Linux im Unternehmen, Heidelberg

Open-Source-Softwareproduktion: Ein neues Innovationsmodell?

MARGIT OSTERLOH, SANDRA ROTA UND BERNHARD KUSTER

1. Einleitung

Während langer Zeit war der Begriff Open Source nur in der Informatikbranche bekannt. Seit Kurzem interessiert sich jedoch auch die Innovationsforschung für das Phänomen. Das ist kein Zufall, handelt es sich hier doch um ein möglicherweise völlig neues Innovationsmodell, das folgende drei Fragen aufwirft (Kogut und Metiu 2001):

1. Warum sind private Eigentumsrechte an Innovationen nicht immer effizient?
2. Warum tragen Menschen freiwillig zu einem öffentlichen Gut bei?
3. Wie muss die Kooperation zwischen kommerziellen Firmen und nicht-kommerziellen Gemeinschaften ausgestaltet sein?

Die Beantwortung dieser Fragen wirft zugleich Licht auf die Bedingungen, unter denen die Produktionsweise von Open-Source-Software auch auf andere Bereiche übertragen werden kann. Schon heute gibt es Projekte, die nach ähnlichen Prinzipien funktionieren (Barbera 1999, Benkler 2002). Beispiele sind die NASA Clickworkers (ein Projekt, bei dem Freiwillige Krater auf dem Mars klassifizieren), Slashdot (ein Forum zum Kommentieren und Klassifizieren von Artikeln) und das Projekt Gutenberg (Scannen und Korrekturlesen von Büchern, deren Urheberrechte abgelaufen sind). Gemeinsames Charakteristikum dieser Projekte ist die freiwillige kollektive Innovation unter weitgehendem Verzicht auf private geistige Eigentumsrechte. Ähnlich wie in weiten Bereichen der wissenschaftlichen Produktion tauschen die Mitglieder dieser virtuellen Gemeinschaften untereinander Beiträge aus, ändern oder verbessern sie, ohne Lizenzverträge abschließen zu müssen.

Im Folgenden werden wir die eingangs gestellten Fragen am Beispiel von Open Source beleuchten. In *Abschnitt zwei* zeigen wir, inwiefern das Open-Source-Modell vom traditionellen ökonomischen Innovationsmodell abweicht. Der Hauptunterschied besteht darin, dass bei der Open-Source-Softwareproduktion unentgeltlich zu einem öffentlichen Gut beigetragen wird. In *Abschnitt drei* bilden wir eine Typologie von Akteuren in der Open-Source-Szene anhand ihrer Motivation, zum öffentlichen Gut beizutragen. In *Abschnitt vier* zeigen wir, dass die „Tragödie der Allmende" bei der Produktion von Open-Source-Software auf zwei Ebenen überwunden werden muss. In *Abschnitt fünf* fragen wir, unter welchen Bedingungen das Open-Source-Modell generalisiert werden kann. Wir unterscheiden dazu motivationale, situative und institutionelle Faktoren. Ihr Zusammenspiel ermöglicht eine erfolgreiche Zusammenarbeit ohne die Bedingungen, die bisher als die Voraussetzung erfolgreicher Innovation galten: Privates geistiges Eigentum der einzelnen Akteure oder die zentrale Autorität eines Unternehmers. Wir kommen zum Ergebnis,

dass das neue Innovationsmodell nur dann verstanden werden kann, wenn man motivationale Faktoren, insbesondere die Dynamik von intrinsischer und extrinsischer Motivation, in die Betrachtung einbezieht.

2. Open-Source-Software: Die Überwindung der „Tragödie der Allmende“

Gemäss dem konventionellen ökonomischen Innovationsmodell entstehen Innovationen nur dann, wenn sie durch starke private Eigentumsrechte (z.B. Patente) geschützt werden (zum Beispiel North 1981). Ohne diese Eigentumsrechte haben Innovationen den Charakter eines öffentlichen Gutes. Öffentliche Güter zeichnen sich dadurch aus, dass niemand von ihrer Nutzung ausgeschlossen werden kann, auch diejenigen nicht, die zur Erstellung des Gutes nichts beigetragen haben (Snidal 1979). Deshalb sind die Anreize, in Innovationen zu investieren, mangelhaft. Es kommt zu einer „Tragödie der Allmende“ (Hardin 1968), d. h. zu einer Übernutzung oder Unterversorgung von Ressourcen der Innovation. Die Folge ist, dass das öffentliche Gut nicht erzeugt wird. Staatlich geschützte intellektuelle Eigentumsrechte sind in diesem Denkmodell unentbehrlich, da sie die Nutzungsrechte an Innovationen regeln und somit die Innovationsanreize stärken.

Es gibt aber auch kritische Stimmen, die die innovationsfördernde Wirkung von intellektuellen Eigentumsrechten anzweifeln (vgl. Bessen und Maskin 2000, Gallini und Scotchmer 2001, Heller und Eisenberg 1998, Mazzoleni und Nelson 1998). Heller und Eisenberg (1998) argumentieren, dass private intellektuelle Eigentumsrechte eine effiziente Weiterentwicklung von Innovationen geradezu verhindern können. Sie bezeichnen dieses Problem als „Tragödie der Anti-Allmende“. Eine „Tragödie der Anti-Allmende“ entsteht, wenn die Eigentumsrechte an einer Ressource auf so viele Parteien aufgesplittert sind, dass eine effiziente Bündelung der Eigentumsrechte praktisch unmöglich gemacht wird. Die Folge ist, dass diese Ressource unternutzt wird. Konkret bedeutet dies: Muss ein potentieller Innovator mit zu vielen Parteien Lizenzverhandlungen führen, können die entstehenden Transaktionskosten derart hoch werden, dass sich die Weiterentwicklung einer Technologie nicht mehr lohnt.

Empirische Evidenz für eine „Tragödie der Anti-Allmende“ in der Softwarebranche finden Bessen und Maskin (2000). Seit den 1980er Jahren wurde der Patentschutz für Software in den USA erheblich gestärkt. Im Gegensatz zu den Voraussagen des klassischen ökonomischen Innovationsmodells stiegen die Ausgaben für Forschung und Entwicklung jedoch nicht an, sondern nahmen relativ zum Umsatz sogar ab.

Die Ursprünge von Open Source liegen in den 1980er Jahren und sind in der kritischen Haltung vieler Softwareentwickler gegenüber Softwarepatenten zu suchen (Lerner und Tirole 2002a). Open Source ist ein Sammelbegriff für Softwarelizenzen, die den Softwarebenutzern nicht nur das Recht einräumen, den Quellcode zu lesen, sondern diesen auch zu verändern und diese Veränderungen zusammen mit dem originalen oder dem veränderten Quellcode wiederum Dritten zugänglich zu machen. Außerdem dürfen keinerlei Lizenzgebühren oder andere Beiträge für die Soft-

ware erhoben werden (O'Reilly & Associates 1999). Allen Open-Source-Lizenzen ist gemein, dass einmal unter eine solche Lizenz gestellter Quellcode nicht reappropriiert werden kann. Das bedeutet, dass das öffentliche Gut nicht in ein privates Gut zurückverwandelt werden darf. Allerdings enthalten die verschiedenen Lizenzen unterschiedlich restriktive Bedingungen bezüglich Weiterentwicklungen des Quellcodes. So erlaubt es beispielsweise die „Birkley Software Distribution License", Weiterentwicklungen unter eine proprietäre Lizenz zu stellen. Andere Lizenzen, wie beispielsweise die GNU General Public Licencse, die auch unter dem Begriff „copyleft" zusammengefasst werden, verbieten dies. Diese Programme müssen in ihrer Gesamtheit unter die gleiche Open-Source-Lizenz gestellt werden.

Open-Source-Programmierer verzichten somit freiwillig auf private Eigentumsrechte an ihren Entwicklungen. Sie bilden virtuelle „Communities of Practice"[1], deren Mitglieder gleichzeitig Nutzer und Entwickler sind, und die selbst entscheiden, wieviel und was sie zu den Projekten beitragen wollen. Die Beiträge werden den anderen Mitgliedern zur Verwendung und Weiterentwicklung zur Verfügung gestellt. Die dadurch erreichte Gleichzeitigkeit von Design und Test der Produkte in raschen Feedback-Zyklen ist charakteristisch für Open-Source-Projekte und ermöglicht eine effiziente Entwicklung zuverlässiger Produkte (Raymond 2001). Die entstehenden Programme stellen öffentliche Güter im klassischen Sinne dar. Weshalb aber beobachten wir hier keine „Tragödie der Allmende"?

Bei der Produktion von Open-Source-Software besteht kein Problem der *Übernutzung*, da keine Rivalität im Konsum vorhanden ist. Zusätzliche Benutzer von Software können sogar positive externe Netzwerkeffekte generieren, etwa indem sie deren Bekanntheitsgrad erhöhen. Schwieriger ist zu erklären, warum das Problem der *Unterversorgung* nicht auftritt. Zu diesem Zweck fragen wir im folgenden Abschnitt, aus welchen Gründen sich Open-Source-Programmierer auch ohne private intellektuelle Eigentumsrechte in diesen Projekten engagieren.

3. Typologie von Open-Source-Programmierern

Es gibt zwei alternative Erklärungen, warum Programmierer freiwillig zu einem öffentlichen Gut beitragen:

- Sie sind eigennützige, kalkulierende Individuen, die rational in ihre Reputation investieren (Lerner und Tirole 2002a) oder die aus der Anpassung der Software an ihre eigenen Bedürfnisse einen Nutzen ziehen, der ihre Kosten übersteigt (von Hippel 2001).
- Sie haben Spaß am Programmieren oder sind überzeugt, zu einer „guten Sache" beizutragen (z.B. Kollock 1999, Raymond 2001, Stallman 1999, Torvalds 1998).

Die beiden alternativen Erklärungen stellen auf zwei unterschiedliche Arten von Motivation ab (Deci und Ryan 2000, Frey 1997, Osterloh und Frey 2000). Die erste bezieht sich auf eine extrinsische, die zweite auf eine intrinsische Motivation der

[1] „Communities of Practice" sind informale Gruppen, die durch geteilte Interessen und Expertise zusammengehalten werden (Brown und Duguid 1991, 1998). Virtuelle „Communities of Practice" wurden beschrieben von Faraj und Wasko (2001) sowie Tuomi (2000).

Beitragenden. Bei der extrinsischen Motivation geht es um eine indirekte Bedürfnisbefriedigung. Die Tätigkeit wird nicht um ihrer selbst willen ausgeführt, sondern zum Beispiel, um Geld zu verdienen. Erst mit dem Geld wird dann das unmittelbare Bedürfnis erfüllt, etwa eine Urlaubsreise. Intrinsische Motivation hingegen bezieht sich auf eine direkte Bedürfnisbefriedigung. Eine Aktivität wird um ihrer selbst willen geschätzt, sie wird auch ohne Belohnung oder Bestrafung ausgeführt.[2]

Intrinsische Motivation hat zwei Dimensionen (Lindenberg 2001), die von verschiedenen Forschungsgruppen untersucht wurden.

- Intrinsische Motivation basierend auf der *Freude an einer Tätigkeit* wurde vor allem durch die Forschungsgruppe um Deci (Deci u.a. 1999) untersucht. Diese Art Tätigkeit verursacht ein „Flow-Erlebnis" (Csikszentmihalyi 1975), wie es sich etwa beim Lesen eines guten Romans, beim Musizieren oder beim Bergsteigen einstellen kann. Dabei bereitet die Tätigkeit selbst einen Genuss.
- Intrinsische Motivation kann aber auch auf die *Erfüllung von Normen um ihrer selbst willen* gerichtet sein. Dies wurde vor allem durch die Forschungsgruppe um Frey (1997) untersucht. Die Relevanz dieser Form der Motivation zeigt sich zum Beispiel bei umweltgerechtem Verhalten, bei Steuerehrlichkeit oder bei kollegialem Verhalten in Unternehmen („organizational citizenship behavior" , vgl. Organ 1988).

Wir werden zeigen, dass in der Open-Source-Softwareproduktion nicht nur alle erwähnten Formen der Motivation auftreten, sondern dass nur das gleichzeitige Vorhandensein dieser verschiedenen Motivationen den durchschlagenden Erfolg einiger Projekte erklären kann. Im Folgenden unterscheiden wir fünf Idealtypen auf Grund ihrer unterschiedlichen Beitragsmotivation, die sich in der Realität allerdings durchaus überlappen können (Hars und Ou 2002, Lakhani u.a. 2002).[3]

3.1. Kommerzielle Unternehmen

Es existieren verschiedene Business-Modelle dafür, wie kommerzielle Unternehmen im Umfeld von Open Source einen Profit erwirtschaften können (einen Überblick geben Berlecon Research 2002 und Markus u.a. 2000). So verkauft etwa der prominenteste kommerzielle Dienstleister Red Hat nicht die Software an sich. Diese kann als öffentliches Gut gratis im Internet heruntergeladen werden. Vielmehr verkauft Red Hat Support-Dienstleistungen rund um die Software. Darüber hinaus integriert das Unternehmen verschiedene Open-Source-Komponenten zu einem vollständigen und verlässlichen Betriebssystem, welches auch von unerfahrenen Benutzern installiert werden kann. Mit diesem Business Modell konnte Red Hat etwa 50% des Marktes für Linux gewinnen (vgl. Deutsche Bank Research 2002). Andere kom-

[2] In der Ökonomie wird intrinsische Motivation kaum betrachtet. Zu den Ausnahmen gehören Akerlof (1982), Benabou und Tirole (2002), Frey (1997) oder Kreps (1997). Milgrom und Roberts (1992, S. 42) stellen fest, dass die Annahme ausschliesslich extrinsisch motivierter Individuen eine „extreme Karikatur" sei. Ihrer Meinung nach müssen Institutionen dennoch so gestaltet werden, als seien alle Individuen nur eigennützig. Dies kann aber zur Verdrängung der intrinsischen Motivation führen, wie später noch ausgeführt wird.

[3] Ein Idealtyp ist nach Weber (1973) ein Konstrukt, bei dem ein Bündel von Merkmalen herausgelöst wird und jene betont werden, die für einen bestimmten Bedeutungszusammenhang als relevant angesehen werden.

merzielle Firmen tragen zu Open-Source-Projekten bei, um damit den Absatz komplementärer Produkte zu fördern. So stellt zum Beispiel Hewlett Packard Druckertreiber zur Verfügung, um ihre Drucker mit Open-Source-Software kompatibel zu machen.

Diese kommerziellen Unternehmen sind unerlässlich für die Verbreitung von Open-Source-Software. Sie machen diese Software, die ursprünglich von Experten für Experten entwickelt wurde, auch für unerfahrene Nutzer handhabbar (Kogut und Metiu 2000).

3.2. Entwickler für den Eigenbedarf

Diese Benutzergruppe passt die Open-Source-Software an ihre eigenen Bedürfnisse an (von Hippel 1988). Sie geben ihre Weiterentwicklungen weiter, wenn ihr persönlicher Nutzen aus der Veröffentlichung ihre Kosten übersteigt.

Wann tritt dieser Fall ein? *Erstens* besteht durch die Veröffentlichung die Chance, dass andere Programmierer den Programmcode unterhalten, weiterentwickeln oder nach Fehlern im Quellcode suchen (z.B. von Hippel 2001, Lerner und Tirole 2002a). *Zweitens* sind die Kosten der Publikation von Beiträgen im Internet sehr niedrig. Deshalb kann sich eine Weitergabe lohnen, auch wenn die Wahrscheinlichkeit von nützlichen Rückmeldungen oder Weiterentwicklungen gering ist.

3.3. Investoren in Reputation

Programmierer können indirekt mit Beiträgen zu Open-Source-Software Geld verdienen, indem sie damit möglichen Arbeitgebern ihre Fähigkeiten signalisieren. Die daraus entstehende Reputation kann dazu führen, dass der Programmierer entweder eine besser bezahlte Anstellung erhält oder einfacher Zugang zu einem Venture-Kapitalisten bekommt. In Open-Source-Software Projekten kann man seine Fähigkeiten einfacher als bei proprietären Produkten signalisieren. Die Mailing-Listen sind öffentlich zugänglich, und oft werden Listen der wichtigsten Beitragenden publiziert (Lerner und Tirole 2002a, Moon und Sproull 2000). Zudem steht es jedem Programmierer frei, seine Beiträge zu signieren. Es ist ein ungeschriebenes Gesetz, dass diese Signaturen nicht entfernt werden dürfen (Raymond 2001). Insoweit ist Open-Source-Softwareproduktion mit der wissenschaftlichen Forschung vergleichbar. Auch hier entsteht Reputation durch häufiges Zitieren, und es gehört zu den Spielregeln wissenschaftlichen Arbeitens, Quellen angemessen anzugeben.

3.4. Homo ludens

Kommerzielle Unternehmen, Entwickler für den Eigenbedarf und Investoren in Reputation gehören zur Klasse der extrinsisch Motivierten. Es gibt jedoch empirische Evidenz dafür, dass viele Programmierer intrinsisch motiviert sind. So stellt Raymond (2001) fest: „We're proving not only that we can do better software, but that joy is an asset"[4]. Diese Aussage steht im Einklang mit Huizingas Bild vom Menschen als *Homo ludens*, dem verspielten Wesen, das Aktivitäten einfach aus Spaß an der Tätigkeit ausführt (Huizinga 1986). Viele Akteure in der Open-Source-Gemeinschaft betonen, dass ihr wichtigstes Motiv Freude am Programmieren ist, wel-

[4] Siehe auch Brooks 1995 und Torvalds 1998.

ches ein „Flow-Erlebnis“ (Csikszentmihalyi 1975) verursacht. Über 70% der Open-Source-Entwickler sagen aus, dass sie während des Programmierens die Zeit vergessen (Lakhani u.a. 2002). In diesem Falle stellen Beiträge zum Open-Source-Code keine Kosten, sondern Nutzen dar. Sie sind nicht Investition, sondern Konsum.

3.5. Reziprokateure

Im Gegensatz zu der Austauschkultur, wie man sie üblicherweise auf Märkten findet, wird die Kultur der Open-Source-Gemeinschaft oft als Geschenkkultur charakterisiert (z.B. Raymond 2001). Ein Geschenk zeichnet sich dadurch aus, dass dem Schenkenden außer der psychologischen Befriedigung keine persönlichen Vorteile erwachsen (Rose-Ackerman 1998). Der Schenkende ist intrinsisch motiviert, die Norm der generalisierten Reziprozität um ihrer selbst willen zu verfolgen. Generalisierte Reziprozität unterscheidet sich von Austauschbeziehungen dadurch, dass es nicht um eine kalkulierende Haltung des „Wie du mir, so ich dir“ geht. Diese würde voraussetzen, dass Geber und Empfänger wechselseitig identifizierbar wären. Dies ist in der Open-Source-Gemeinschaft jedoch typischerweise nicht der Fall. Normen generalisierter Reziprozität führen dazu, dass Menschen in einer Gruppe einander Hilfe zukommen lassen, ohne dass einzelne Gruppenmitglieder identifiziert werden (Constant u.a. 1996). Eine solche Motivation drückt sich durch die folgende Aussage aus: “The person I help may never be in the position to help me, but someone else might be“ (Rheingold 1993). Teilnehmer einer Newsgroup über technische Computerprobleme berichten, dass sie der Gemeinschaft, von der sie Hilfe erhalten haben, etwas zurückgeben wollen (Faraj und Wasko 2001). Wenn sie Fragen beantworten, richtet sich ihre Hilfe nicht an den Fragesteller allein, sondern an die Gruppe als Ganzes (Wellman und Gulia 1999). Das Wohl der Gemeinschaft fließt in die Präferenzen der Einzelnen ein. Lakhani und von Hippel (2003) fanden in einer empirischen Studie über Apache Newsgroups, dass generalisierte Reziprozität („I have been helped before, so I reciprocate“, „I help now so I will be helped in the future“) sogar das am meisten genannte Motiv ist.

Wichtig ist dabei, dass Reziprokateure auf zwei Ebenen zu öffentlichen Gütern beitragen.[5] Sie fördern ein öffentliches Gut erster Ordnung, indem sie die Funktionalität und Qualität der Software direkt verbessern. Zum öffentlichen Gut zweiter Ordnung tragen sie bei, indem sie dafür sorgen, dass der Quellcode ein öffentliches Gut bleibt. Dabei gibt es durchaus verschiedene Meinungen darüber, welche Lizenzart am besten geeignet ist, die Entwicklung und Verbreitung von Open-Source-Software voranzutreiben. Während einige die Ansicht vertreten, dass nur die GNU General Public License die Offenheit des Quellcodes garantiere, sind andere der Ansicht, der virale Effekt dieser Lizenz schränke die Freiheit der Programmierer zu sehr ein.[6]

---

[5] Eine begriffliche Unterscheidung zwischen öffentlichen Gütern erster und zweiter Ordnung wird im nächsten Abschnitt vorgenommen.

[6] „Viral“ bezieht sich auf die Eigenschaft der GPL, dass ein Programm, das unter der GPL lizenzierten Code enthält, in seiner Gesamtheit unter die GPL gestellt werden muss. Der GPL Code ist in diesem Sinne ansteckend.

4. Zwei Ebenen der „Tragödie der Allmende“: Soziale Dilemmata erster und zweiter Ordnung

Im Folgenden legen wir dar, dass die „Tragödie der Allmende“ bei der Produktion von Open-Source-Software auf zwei Ebenen überwunden werden muss.

Die von Hardin (1968) beschriebene „Tragödie der Allmende“ gehört zu den sogenannten sozialen Dilemmata. Diese entstehen, wenn Handlungen von eigennützigen Akteuren nicht zu kollektiv erwünschten Resultaten führen (Dawes 1980, Ostrom 1998). Im Gegensatz zu Hardins pessimistischer Voraussage sind allerdings längst nicht alle Menschen rein eigennützig. Empirische Untersuchungen zeigen vielmehr, dass viele Menschen freiwillig zu öffentlichen Gütern beitragen (z.B. Frey und Meier 2002), etwa weil sie Freude an der Tätigkeit haben oder weil sie einen Nutzen daraus ziehen, wenn das Gemeinschaftswohl erhöht wird. In diesem Fall wird das soziale Dilemma in ein Koordinationsspiel transformiert, bei dem es mehr als ein Gleichgewicht gibt (Sen 1974). Soziale Dilemmata können demnach gelöst werden, wenn es eine genügende Anzahl von intrinsisch motivierten Akteuren gibt.

Zu beachten ist dabei, dass soziale Dilemmata auf zwei Ebenen entstehen können. Auf der *ersten* Ebene geht es um das Trittbrettfahren bei der Produktion des Quellcodes: Weil dieser im Fall der Open-Source-Software ein öffentliches Gut ist und niemand von der Nutzung ausgeschlossen werden kann, besteht das Problem der Unterversorgung.

Auf der *zweiten* Ebene geht es um die Sanktion von Regelverletzern bei der Erstellung des öffentlichen Gutes auf der ersten Ebene. Die schlimmsten Regelverletzungen bei der Open-Source-Softwareproduktion sind die Missachtung von Lizenzbestimmungen, der Gebrauch von Open-Source-Komponenten in kommerziellen Produkten, ohne der Gemeinschaft etwas zurückzugeben, und die Entfernung der Signaturen (der „credits“) von Programmteilen.[7] Die Sanktionierung solcher Regelverletzungen ist selbst ein öffentliches Gut: „Punishment almost invariably is costly to the punisher, while the benefits from punishment are diffusely distributed over all members. It is, in fact, a public good“ (Elster 1989, S. 41). Deshalb entsteht ein soziales Dilemma höherer Ordnung. Als klassische Lösung dieses Dilemmas gilt das Einsetzen einer zentralen Instanz, welche die Einhaltung der Regeln überwacht und dafür ein Entgelt erhält (Alchian und Demsetz 1972). Wenn es jedoch eine genügende Anzahl Freiwilliger gibt, denen es ein Anliegen ist, dass Regeln eingehalten werden, und die sogar unter Inkaufnahme persönlicher Nachteile bereit sind, Regelbrecher zu sanktionieren, kann das soziale Dilemma zweiter Ordnung ohne eine zentrale Autorität gelöst werden. Es gibt zahlreiche Laborexperimente, die zeigen, dass es solche Personen gibt (Camerer und Fehr 2003, Ledyard 1995). In der Open-Source-Gemeinschaft funktioniert Sanktionierung zum Beispiel durch „flaming“, das öffentliche Beschimpfen von Personen im Internet. Flaming ist nicht nur ein Sanktionsmittel, sondern hat zugleich eine expressive Funktion. Es demonstriert, dass eine zentrale Norm der Gemeinschaft verletzt wurde (Kollock und Smith 1996). Experimente zeigen, dass eine solche Demonstration die Bereitschaft, zu öffentli-

[7] Vgl. zum Beispiel Raymond (2001), der feststellt: „Surreptitiously filing someone's name off a project is, in cultural context, one of the ultimate crimes.“

chen Gütern beizutragen, signifikant erhöht (Sally 1995). Andere Sanktionsmöglichkeiten sind das „kill-filing", d. h. die öffentliche Bekanntgabe, dass man von einer Person keine E-Mails mehr erhalten möchte, und das „shunning", d. h. das Ignorieren von E-Mails.

Die Regelüberwachung ist in der Open-Source-Softwaregemeinschaft im Allgemeinen einfach, weil das Internet eine sehr hohe Transparenz gewährleistet.[8] Hingegen ist die Sanktionierung von Missetätern weitaus problematischer. *Erstens* sind die genannten Sanktionen weitgehend informeller Natur. Den Regelbrechern können keine direkten monetären Kosten auferlegt werden. *Zweitens* sind die Mitglieder der Gemeinschaft teilweise anonym, und es bestehen keine klar identifizierbaren Gruppen- und Ressourcengrenzen. Damit sind die von Ostrom (1990) ermittelten Bedingungen einer Selbstregulation in Gruppen nicht gegeben. Dennoch scheint die Selbstregulation in der Open-Source-Gemeinschaft zu funktionieren. Es zeigt sich, dass Sanktionen einen signifikanten Effekt auf das Verhalten haben (Kollock und Smith 1996), obwohl diese Sanktionen oft nur Scham bei den Missetätern erzeugen können. Scham entfaltet aber nur dann eine Wirkung, wenn auch bei den Sanktionierten eine intrinsische Motivation vorhanden ist. Rein extrinsisch motivierte Egoisten fühlen keine Scham (Elster 1999, Orr 2001).

Damit sollte deutlich geworden sein, dass intrinsische Motivation eine Voraussetzung dafür ist, dass öffentliche Güter erster und zweiter Ordnung in einer Gemeinschaft von Freiwilligen entstehen können, und somit eine Bedingung für das Funktionieren des Open-Source-Innovationsmodells darstellt. Im Folgenden werden wir der Frage nachgehen, wie diese intrinsische Motivation erhalten werden kann. Dazu unterscheiden wir einen motivationalen, einen situativen und einen institutionellen Faktor.

5. Bedingungen für das Funktionieren des Open-Source-Innovationsmodells: Drei Faktoren

Open-Source-Entwickler bilden nicht die einzigen freiwilligen Gemeinschaften, bei denen das geistige Eigentum der Gemeinschaft gehört und die ohne zentrale Instanz auskommen (Benkler 2002). Dennoch zeichnet sich die große Mehrheit aller Innovationsprojekte immer noch durch das Bestehen privater geistiger Eigentumsrechte aus.

In diesem Abschnitt gehen wir näher darauf ein, unter welchen Bedingungen ein Innovationsmodell ohne geistige Eigentumsrechte funktionieren kann, ohne dass es zu einer „Tragödie der Allmende" kommt.

Zu diesem Zweck gehen wir zunächst nochmals kurz auf die motivationalen Bedingungen für den Erfolg des Open-Source-Innovationsmodells ein.

Wir zeigen, dass intrinsische und extrinsische Motivation in Open-Source-Projekten nicht nur koexistieren, sondern unter bestimmten situativen und institutionellen Bedingungen komplementär sind. Anschließend stellen wir diese situativen und institutionellen Bedingungen dar.

[8] Dies mit einer Ausnahme: Die illegale Integration von Open-Source-Elementen in proprietäre Software ist nur schwer nachweisbar, da bei proprietärer Software der Quellcode nicht zugänglich ist.

5.1. Der motivationale Faktor: Portfolio intrinsischer und extrinsischer Motivation

Nur durch das Engagement intrinsisch motivierter Akteure entwickeln Open-Source-Projekte genügend Zugkraft, um extrinsisch motivierte Beitragende anzuziehen (Bessen 2002, Franck und Jungwirth 2002):

- Kommerzielle Unternehmen können erst Geld mit ihren Komplementärprodukten verdienen, wenn die Software bereits eine gewisse Reife und Verbreitung erreicht hat.
- Entwickler für den Eigenbedarf haben erst dann einen die Kosten übersteigenden Nutzen, wenn die Entwicklung der Software bereits weit fortgeschritten ist und funktionsfähige Produkte vorliegen.
- Für Investoren in Reputation lohnt sich der Einstieg erst bei bereits bekannten und erfolgreichen Projekten, die genügend öffentliche Aufmerksamkeit erhalten.

Der Erfolg von Open-Source-Software ist aber auch abhängig von eigennützigen, extrinsisch motivierten Akteuren. So haben kommerzielle Unternehmen, Entwickler für den Eigenbedarf und Investoren in Reputation eine wichtige Funktion für die Verbreitung von Open-Source-Software: Sie passen die Produkte, die ursprünglich von Experten für Experten entwickelt wurden, an die Bedürfnisse einer breiten Benutzerschaft an.

Wir wissen noch wenig über das zahlenmäßige Verhältnis von intrinsisch und extrinsisch motivierten Entwicklern. Auf Grund verschiedener empirischer Untersuchungen (Lakhani und von Hippel 2003, Ghosh u.a. 2002, Hars und Ou 2002, Lakhani u.a. 2002) kann man vorläufig die Schlussfolgerung ziehen, dass extrinsisch und intrinsisch motivierte Akteure jeweils etwa die Hälfte stellen.

5.2. Der situative Faktor: Niedrige Kosten

Wenn das Open-Source-Innovationsmodell für die Entwicklung von Software so gut funktioniert, weshalb wird es dann nicht überall, beispielsweise in der pharmazeutischen Industrie, angewandt? Die einfache Antwort lautet, dass dieses Innovationsmodell nur in Situationen brauchbar ist, in denen der Nutzen der Entwickler deren Kosten übersteigt. Dies schließt auch den intrinsisch motivierten Nutzen ein. Auch unter intrinsisch motivierten Individuen sind Märtyrer und Heilige nur selten anzutreffen. Wohltäter tragen umso mehr zu einem öffentlichen Gut bei, je niedriger die Kosten sind (Rose-Ackerman 2001, S. 553). Es gilt: Je mehr Kosten moralisches Verhalten verursacht, desto seltener ist es (North 1990, S. 43). Sind die Kosten hingegen niedrig, sind viele bereit, ihren Beitrag zum öffentlichen Gut zu leisten (Kirchgässner 1992, Kliemt 1986), der in seiner Gesamtheit – wie im Fall Open Source – hoch sein kann.

Kosten wie auch Nutzen können in zwei Aspekte aufgeteilt werden, auf die wir im Folgenden detaillierter eingehen wollen: Kosten und Nutzen der Produktion von Quellcode sowie Kosten und Nutzen der Publikation dieses Quellcodes.

- Softwareentwicklung zeichnet sich durch ihren sequentiellen und komplementären Charakter aus (Bessen und Maskin 2000). Dadurch werden die

Kosten bei der Produktion des Quellcodes für den einzelnen Beitragenden gering gehalten.

- *Sequentiell* bedeutet, dass Innovation auf vorhergehenden Entwicklungen aufbaut, sodass Innovation nicht in radikalen, großen Schritten, sondern inkrementell erfolgt.[9]
- Innovation ist *komplementär*, wenn durch die Teilnahme vieler Entwickler die Chance steigt, eine Lösung für das gemeinsame Problem zu finden, weil Synergien zwischen den verschiedenen Lösungsansätzen auftreten.
- Die monetären Kosten der Publikation des Quellcodes sind im Verhältnis zum potentiellen Nutzen niedrig. Zwei Arten von Kosten müssen betrachtet werden. Zum Ersten sind die Kosten der Verbreitung gering. Beiträge zu Open-Source-Projekten können einfach auf der entsprechenden Internetseite publiziert werden. Zweitens sind die Kosten aus dem Verlust von privatem intellektuellem Eigentum oft niedrig, da viele Entwickler gar nicht die Möglichkeit hätten, ihr Produkt zu vermarkten, oder der Beitrag für eine Vermarktung zu geringfügig ist (z.B. im Falle einer reinen Fehlerbeseitigung). Der erwartete Nutzen, bestehend aus dem Feedback oder möglichen Weiterentwicklungen durch Dritte, ist somit oft größer als die direkten Kosten.

Auch wenn die Kosten für jeden einzelnen Entwickler niedrig sind, bedeutet dies nicht, dass auf intrinsische Motivation verzichtet werden kann. In Abschnitt vier haben wir gezeigt, dass eine nachhaltige Lösung der sozialen Dilemmata erster und zweiter Ordnung ohne intrinsisch motivierte Akteure nicht möglich ist. Daher ist die Frage von großer Bedeutung, wie intrinsische Motivation erhalten oder gefördert werden kann.

5.3. Der institutionelle Faktor: Intrinsische Motivation darf nicht verdrängt werden

Wir wissen noch wenig darüber, wie intrinsische Motivation entsteht. Mehr ist über die Bedingungen bekannt, unter denen eine vorhandene intrinsische Motivation verdrängt oder verstärkt wird (Frey und Osterloh 2002):

- Intrinsische Motivation ist von der empfundenen Selbstbestimmung abhängig. Deshalb wird sie durch externe Interventionen gemindert, die als kontrollierend empfunden werden. Ein Verstärkungseffekt tritt ein, wenn das Selbstbestimmungsgefühl durch Autonomie erhöht wird (Deci und Ryan 2000, für einen umfassenden Überblick über die empirische Evidenz siehe Frey und Jegen 2001).

  In Open-Source-Projekten wird das Gefühl der Selbstbestimmung auf zwei Arten gestärkt. Erstens entscheiden die Programmierer selbst, bei welchem Projekt sie mitmachen und welche Beiträge sie leisten (Benkler 2002, für eine empirische Untersuchung siehe von Krogh u.a. 2003). Zweitens existiert eine Vielzahl von Partizipationsmöglichkeiten. Zahlreiche empirische Labor- und Felduntersuchungen zeigen, dass durch Partizipation die Einsatzbereitschaft

[9] Von Krogh u.a. (2003) untersuchten anhand des Beispieles Freenet den Prozess, den ein Teilnehmer durchlaufen muss, bis er ein vollwertiges Mitglied des Projektes wird. Dieser Prozess beinhaltet langwierige Lernprozesse und zieht sich typischerweise über längere Zeit hin.

für selbstgewählte Projekte gestärkt wird (Frey u.a. 2002, Frey und Stutzer 2002, Osterloh u.a. 2002, Organ und Ryan 1995). Obwohl die Entscheidungsverfahren verschiedener Open-Source-Projekte sehr unterschiedlich sind (für eine Übersicht siehe Markus u.a. 2000), ist ihnen doch gemein, dass sie ein hohes Ausmaß an Partizipation und Autonomie gewährleisten.

- Empirische Evidenz zeigt, dass viele Individuen freiwillig zu einem öffentlichen Gut beitragen, falls auch andere einen Anteil leisten, d.h. sie sind konditional kooperationsbereit (Fischbacher u.a. 2001, Levi 1988, Ostrom 2000).

Daraus folgen zwei Konsequenzen: *Erstens* wird intrinsische Motivation verdrängt, wenn es zu viele Trittbrettfahrer gibt. Institutionelle Arrangements müssen sicherstellen, dass dieses Verhalten verhindert wird. Open-Source-Lizenzen, insbesondere „copyleft"-Lizenzen wie die GNU General Public License, sind Beispiele für solche institutionellen Arrangements (Franck und Jungwirth 2002).[10] Die virale[11] Eigenschaft der „copyleft"-Lizenzen unterstellt Open-Source-Projekte einer besonderen Selbstbeschränkung, genannt „nondistribution constraint" (Hansmann 1980), die typisch für Non-Profit-Organisationen ist (Franck und Jungwirth 2002). Sie gewährleistet, dass niemand die freiwilligen Beiträge oder Spenden in seinen Privatbesitz überführen kann. Auch Organisationen wie das Rote Kreuz oder viele Universitäten unterwerfen sich dieser Beschränkung. Kommerzielle Unternehmen, die im Open-Source-Umfeld tätig sind und von Beiträgen freiwilliger Mitarbeiter abhängen, müssen glaubwürdig kommunizieren, dass sie sich nicht einseitig an diesen freiwilligen Beiträgen bereichern, wollen sie die konditionale Kooperationsbereitschaft erhalten. Anderenfalls ist ihr Business-Modell gefährdet (Benkler 2002).

Dies kann am Beispiel von Red Hat gezeigt werden. Red Hat verkauft nicht den eigentlichen Linux Code. Vielmehr verkauft das Unternehmen Support und Dienstleistungen rund um Linux. Dazu werden unter anderem verschiedene Open-Source-Programme auf ihre Kompatibilität getestet und auf einer CD-ROM zusammengestellt. Kurze Zeit nach der Veröffentlichung einer solchen CD-Rom verkauften Konkurrenten Kopien dieser CD-ROM zu einem bedeutend tieferen Preis. Obwohl Red Hat nicht im Besitz des Quellcodes der Open-Source-Programme auf der CD-ROM ist, hat das Unternehmen doch Eigentumsrechte an Teilen der CD-ROM. Wie reagierte Red Hat? Die erstaunliche Antwort lautet: überhaupt nicht. Die Manager von Red Hat argumentierten, dass die Normen der Open-Source-Gemeinschaft die Durchsetzung von Eigentumsrechten nicht zulassen. Weil Red Hat vom Wohlwollen der Open-Source-Gemeinschaft abhängig ist, hat sich das Unternehmen an die Normen der Gemeinschaft gehalten, um die konditionale Kooperationsbereitschaft der freiwilligen Entwickler nicht zu zerstören.

Zweitens wird intrinsische Motivation verdrängt, wenn Kooperationsregeln nicht eingehalten werden. Wie bereits erwähnt, sind auf Grund der Transparenz des Internets die Kosten für Beobachtung und Sanktionierung von Regelbrechern in der

[10] Für Ausführungen zu den verschiedenen Lizenzarten vgl. Abschnitt 2.

[11] Der Begriff „viral" umschreibt plastisch die ansteckende Eigenschaft dieser Lizenzen. Jeder Quellcode, der mit unter einer solchen Lizenz stehendem Code in Berührung kommt, wird mit dem Open-Source-„Virus" infiziert.

Open-Source-Gemeinschaft gering. Diese Sanktionen sind jedoch häufig nur dann wirksam, wenn die Regelbrecher wenigstens ein Minimum an intrinsisch motivierter Scham empfinden. Die in Open-Source-Projekten praktizierten informellen Sanktionen sind besser geeignet, die intrinsisch motivierte Kooperationsbereitschaft zu erhalten, weil sie das Gefühl der Selbstbestimmung weniger mindern als formelle Strafen (Kollock und Smith 1996, Orr 2001, Ostrom 1990).

6. Schlussbemerkungen

Open-Source-Software ist das bekannteste Beispiel eines neuen und erfolgreichen Innovationsmodells, das aber keineswegs einzigartig ist. Welche sind die Bedingungen, unter denen dieses Innovationsmodell generell funktionieren kann? Dazu haben wir drei Fragen aufgeworfen:

Die *erste* Frage lautete, warum private Eigentumsrechte an Innovation manchmal ineffizient sind. Diese Frage nach den situativen Bedingungen haben wir folgendermaßen beantwortet: Private Eigentumsrechte an Wissen sind dann ineffizient, wenn *erstens* die Innovationen sequentiell und komplementär sind, und *zweitens* die Kosten der Offenlegung im Vergleich zum erwarteten Nutzen gering sind. In diesem Fall bestehen auch ohne private intellektuelle Eigentumsrechte Innovationsanreize (vgl. z.B. Bessen und Maskin 2000). Zusätzlich droht eine „Tragödie der Anti-Allmende": Die Inhaber privater intellektueller Eigentumsrechte können Dritte von der Nutzung einer Innovation ausschließen und so deren Weiterentwicklungen behindern. Diese Situation tritt nicht nur bei der Softwareproduktion auf, sondern auch bei anderen wissensintensiven Gütern, die auf Gruppenarbeit angewiesen sind (Benkler 2002). Ungeeignet ist dieses Innovationsmodell dann, wenn hohe Einzelinvestitionen nötig sind, wie zum Beispiel in der pharmazeutischen Industrie. In diesem Fall wären die Kosten der Offenlegung im Vergleich zum erwarteten Nutzen einer kollektiven Weiterentwicklung sehr hoch.

Die *zweite* Frage lautete, warum Individuen freiwillig zu einem öffentlichen Gut beitragen. Diese Frage bezieht sich auf den motivationalen Faktor. Wir haben gezeigt, dass intrinsisch und extrinsisch motivierte Individuen hierbei effizient interagieren. *Intrinsisch* motivierte Personen tragen zum öffentlichen Gut erster und zweiter Ordnung bei. Das öffentliche Gut erster Ordnung bezieht sich auf die Produktion von Open-Source-Software. Das öffentliche Gut zweiter Ordnung, die Sanktionierung von Regelbrechern, ist notwendig, um die konditionale Kooperationsbereitschaft zu erhalten. Sofern dies nicht nur mit Nutzen – d.h. mit Spaß an der Tätigkeit – verbunden ist, kann man davon ausgehen, dass auch intrinsisch Motivierte um so eher zum öffentlichen Gut beitragen, je geringer die individuellen Kosten sind und je leichter sich die einzelnen, kleinen Beiträge zu einem effektvollen Ganzen aufsummieren lassen. Genau dies ermöglicht die Sequentialität und Komplementarität der Softwareproduktion (im Unterschied etwa zur pharmazeutischen Produktion). Deshalb sind der motivationale und der situative Faktor miteinander verbunden. *Extrinsisch* motivierte Individuen tragen aus eigennützigen Gründen zum öffentlichen Gut bei. Mit ihrem Beitrag wollen sie ihre eigenen Produktbedürfnisse befriedigen, in ihre Reputation investieren oder die Benutzerbasis

für ihre Komplementärprodukte erweitern. Wir haben gezeigt, dass der Erfolg des neuen Innovationsmodells das Zusammenspiel von intrinsischer und extrinsischer Motivation voraussetzt.

Die *dritte* Frage bezog sich auf die Kooperation zwischen kommerziellen Untenehmen und nicht-kommerziellen Gemeinschaften. Die Beantwortung dieser Frage ist verbunden mit dem institutionellen Faktor. Lizenzen wie „copyleft" sind ein gutes institutionelles Arrangement, um die konditionale Kooperationsbereitschaft von intrinsisch motivierten Programmierern zu erhalten (Franck und Jungwirth 2002). Kommerzielle Unternehmen – wie etwa Red Hat – müssen die Normen der generalisierten Reziprozität respektieren, auch wenn sie über die rechtlichen Lizenzverpflichtungen hinausgehen, um die Kooperationsbereitschaft der Open-Source-Programmierer nicht zu untergraben. Bisher ist dies der Fall, obwohl immer mehr große kommerzielle Unternehmen (wie z.B. IBM) in der Open-Source-Gemeinde eine gewichtige Rolle spielen. Es bleibt abzuwarten, ob die für das Funktionieren des Open-Source-Innovationsmodells notwendige Balance zwischen intrinsischer und extrinsischer Motivation auch bei einem Überwiegen kommerzieller Akteure in Zukunft aufrechterhalten werden kann.

Literatur

Akerlof, G.A. (1982): *Labor Contracts as Partial Gift Exchange*, Quart. J. of Econom. 97(4), S. 543–569

Alchian, A. / Demsetz, H. (1972): *Production, Information Costs, and Economic Organization*, Am. Econom. Rev. 62, S. 777–795

Barbera, F.R. (1999): *Berkman Center Rebuffed*, Harvard Law Record, 109(9),1

Benabou, R., J. Tirole (2002): *Intrinsic and Extrinsic Motivation*, Arbeitspapier, Princeton University, Princeton, NJ

Benkler, Y. (2002): *Coase's Penguin, or, Linux and the Nature of the FirmK,* The Yale Law J. 112(3)

Berlecon Research (2002): *Floss Final Report – Part 3: Basics of Open Source Software Markets and Business Models*,
online http://www.berlecon.de/studien/floss/FLOSS_Grundlagen.pdf

Bessen, J. (2002): *What Good is Free Software?* In: Hahn, R. W. (Hrsg.): Government Policy toward Open Source Software, Washington, DC, S. 12–33

Bessen, J., E. Maskin (2000): *Sequential Innovation, Patents and Imitation*, Arbeitspapier MIT, Cambridge, MA

Brooks, F. P. (1995): *The Mythical Man-Month: Essays on Software Engineering*, Reading, MA

Brown, J. S. /. Duguid, P. (1991): *Organizational Learning and Communities-of-Practice: Toward a Unified View of Working, Learning, and Innovation*, Organ. Sci. 2(1), S. 40–57

Brown, J. S. / Duguid, P. (1998): *Organizing Knowledge*, California Management Rev. 40(3), S. 90–111

Camerer, C. F. / Fehr E. (2003): *Measuring Social Norms and Preferences using Experimental Games: A Guide for Social Scientists*, In: Henrich, J. / R. Boyd / S.

Bowles / C. Camerer / E. Fehr / H. Gintis (Hrsg.): Cooperation, Self Interest and Punishment: Experimental and Ethnographic Evidence from Small-scale Societies, Oxford, NY, im Druck

Constant, D. / Sproull, L. / Kiesler, S. (1996): *The Kindness of Strangers: The Usefulness of Electronic Weak Ties for Technical Advice*, Organ. Sci. 7(2), S. 119–135

Csikszentmihalyi, M. (1975): *Beyond Boredom and Anxiety*, San Francisco, CA

Dawes, R. M. (1980): *Social Dilemmas*, Annual Rev. of Psych. 31, S. 169–193

Deci, E. L. / Ryan, R. M. (2000): *The „What" and „Why" of Goal Pursuits: Human Needs and the Self-Determination of Behavior*, Psych. Inq. 11(4), S. 227–268

Deci, E. L. / Koestner, R. / Ryan,R. M. (1999): *Meta-Analytic Review of Experiments: Examining the Effects of Extrinsic Rewards on Intrinsic Motivation*, Psych. Bull. 125 (3), S. 627–668

Deutsche Bank Research. (2002): *Free software, Big Business? Open Source Programme erobern Wirtschaft und öffentlichen Sektor*, E-conomics. 32

Elster, J. (1989): *The Cement of Society: A Study of Social Order*, New York, NY

Elster, J. (1999): *Alchemies of the Mind: Rationality and the Emotions*, New York, NY

Faraj, S. / Wasko, M. M. (2001): *The Web of Knowledge: An Investigation of Knowledge Exchange in Networks of Practice*, Arbeitspapier Florida State University, Tallahassee, FL

Fischbacher, U. / Gächter, S. / Fehr, E. (2001): *Are People Conditionally Cooperative? Evidence from Public Good Experiments*, Econom. Lett. 71(3), S. 397–404

Franck, E. / Jungwirth C. (2002): *Reconciling Investors and Donators – The Governance Structure of Open Source*, Arbeitspapier Universität Zürich, Zürich, Schweiz

Frey, B. S. (1997): *Not Just for the Money: An Economic Theory of Personal Motivation*, Cheltenham

Frey, B. S., R. Jegen (2001): *Motivation Crowding Theory: A Survey of Empirical Evidence*, J. of Econom. Surv. 15(5), S. 589–611

Frey, B. S., / Meier, S. (2002): *Pro-social Behavior, Reciprocity or Both?* Arbeitspapier Universität Zürich, Zürich

Frey, B. S. / Osterloh, M. (2002): *Successful Management by Motivation. Balancing Intrinsic and Extrinsic Incentives*, Berlin

Frey, B. S. / Stutzer, A. (2002): *Beyond Outcomes: Measuring Procedural Utility*, Arbeitspapier Universität Zürich, Zürich, Schweiz

Frey, B.S. / Benz, M. / Stutzer, A. (2002): *Introducing Procedural Utility: Not only What, but also How Matters*, Arbeitspapier Universität Zürich, Zürich, Schweiz

Gallini, N. / Scotchmer, S. (2001): *Intellectual Property: When is it the Best Incentive System?* In: Jaffe, A., J. Lerner S. Stern (Hrsg.): Innovation Policy and the Economy Bd. 2. Cambridge, MA, S. 51–78

Ghosh, R. A. / Glott, R. / Krieger, B. / Robles, B. (2002): *Floss Final Report – Part 4: Survey of Developers*,
online http://www.infonomics.nl/FLOSS/report/FLOSS_Final4.pdf

Hansmann, H. B. (1980): *The Role of Nonprofit Enterprise*, Yale Law J. 89, S. 835–901

Hardin, G. (1968): *The tragedy of the commons*, Science 162(3859), S. 1243–1248

Hars, A. / Ou, S. (2002): *Working for Free? Motivations for Participating in Open Source Projects*, Internat. J. of Electron. Comm. 6(3), S. 25–39

Heller, M. A. (1998): *The Tragedy of the Anticommons: Property in the Transition from Marx to Markets*, Harvard Law Rev. 111(3), S. 622–688

Heller, M. A. / Eisenberg, R. S. (1998): *Can Patents Deter Innovation? The Anticommons in Biomedical Research*, Science 280(5364), S. 698–701

Huizinga, J. (1986): *Homo Ludens*, Boston, MA

Kirchgässner, G. (1992): *Toward a Theory of Low-cost-decision*, Eur. J. of Polit. Econom. 8(2), S. 305–320

Kliemt, H. (1986): *The Veil of Insignificance*, Eur. J. of Polit. Econom. 2(3), S. 333–344

Kogut, B., A. Metiu (2000): *The Emergence of E-Innovation: Insights from Open Source Software Development*, Arbeitspapier Reginald H. Jones Center, Philadelphia, PA

Kogut, B., A. Metiu (2001): *Open Source Software Development and Distributed Innovation*, Oxford Rev. of Econom. Policy. 17(2), S. 248–264

Kollock, P. (1999): *The Economics of Online Cooperation: Gifts and Public Goods in Cyberspace*, in: Smith, M. A. / Kollock, P. (Hrsg.): Communities in Cyberspace. London, S. 220–242

Kollock, P. / Smith, M. (1996): *Managing the Virtual Commons: Cooperation and Conflict in Computer Communities*, in: Herring, S. (Hrsg.): Computer-mediated Communication: Linguistic, Social, and Cross-cultural Perspectives. Amsterdam, S. 109–128

Kreps, D. M. (1997): *Intrinsic Motivation and Extrinsic Incentives*, Amer. Econom. Rev. 87(2), S. 359–364

Lakhani, K. / Hippel, E. von (2003): *How Open Source Software Works: „Free" User-to-User Assistance*, Res. Policy. 32(7), S. 923–943

Lakhani, K. / Wolf, B. / Bates, J. / DiBona, C. (2002): *The Boston Consulting Group Hacker Survey*,
online http://www.osdn.com/bcg/BCGHACKERSURVEY.pdf and http://www.osdn.com/bcg/BCGHACKERSURVEY-0.73.pdf

Ledyard, J. (1995): *Public Goods: A Survey of Experimental Research*, in: Roth, A. / Kagel, J. (Hrsg.): Handbook of Experimental Economics, Princeton, NJ, 111–194

Lerner, J. / Tirole, J. (2002a): *Some Simple Economics of Open Source*, J. of Indust. Econom. 50(2), S. 197–234

Lerner, J., J. Tirole. (2002b): *The Scope of Open Source Licensing*, Arbeitspapier Harvard Business School, Boston, MA

Levi, M. (1988): *Of Rule and Revenue*, Berkeley, CA

Lindenberg, S. (2001): *Intrinsic Motivation in a New Light*, Kyklos. 54(2/3), S. 317–343

March, J. G. (1991): *Exploration and Exploitation in Organizational Learning*, Organ. Sci. 2(1), S. 71–87

Markus, M. L. / Manville, B. / Agres, C. E. (2000): *What Makes a Virtual Organization Work?* Sloan Management Rev. 42(1), S. 13–26

Mazzoleni, R. / Nelson, R. R. (1998): *The Benefits and Costs of Strong Patent Protection: A Contribution to the Current Debate*, Res. Policy. 27(3), S. 275–286

Merton, R. K. (1983): *Auf den Schultern von Riesen : Ein Leitfaden durch das Labyrinth der Gelehrsamkeit*, Frankfurt am Main

Milgrom, P. R. / Roberts, J. (1992): *Economics, Organization and Management*, Englewood-Cliffs, NJ

Moon, J. Y. / Sproull, L. (2000): Essence of Distributed Work: The Case of the Linux Kernel. Firstmonday, 5(11)

North, D. C. (1981): *Structure and Change in Economic History*, New York, NY

North, D. C. (1990): *Institutions, Institutional Change, and Economic Performance*, New York, NY

O'Reilly & Associates, Inc. (1999): *Open Source: kurz & gut*, Köln

Organ, D. W. (1988): *Organizational Citizenship Behavior: The Good Soldier Syndrome*, Lexington, MA

Organ, D. W. / Ryan, K. (1995): *A Meta-analytic Review of Attitudinal and Dispositional Predictors of Organizational Citizenship Behavior*, Pers. Psych. 48(4), S. 776–801

Orr, S. W. (2001): *The Economics of Shame in Work Groups: How Mutual Monitoring Can Decrease Cooperation in Teams*, Kyklos. 54(1), S. 49–66

Osterloh, M., B. / Frey, B. S. (2000): *Motivation, Knowledge Transfer, and Organizational Forms*, Organ. Sci. 11(5), S. 538–550

Osterloh, M. / Frost, J. / Frey, B. S. (2002): *The Dynamics of Motivation in New Organizational Forms*, International Journal of the Economics of Business, Special Issue on New Organizational Forms. 9(1), S. 61–77

Ostrom, E. (1990): *Governing the Commons: The Evolution of Institutions for Collective Action*, New York, NY

Ostrom, E. (1998): *A Behavioral Approach to the Rational-Choice Theory of Collective Action*, Amer. Polit. Sci. Rev. 92(1), S. 1–22

Ostrom, E. (2000): *Crowding out Citizenship*, Scand. Polit. Stud. 23(1), S. 3–16

Raymond, E. S. (2001): *The Cathedral and the Bazaar*, Sebastopol, CA

Rheingold, H. (1993): *The Virtual Community: Homesteading on the Electronic Frontier*, New York, NY

Rose-Ackerman, S. (1996): *Altruism, Nonprofits, and Economic Theory*, J. of Econom. Lit. 34(2), S. 701–728

Rose-Ackerman, S. (1998): *Bribes and Gifts*, in: Ben-Ner, A. / Putterman, L. (Hrsg.): Economics, Values, and Organization, New York, NY, S. 296–328

Rose-Ackerman, S. (2001): *Trust, Honesty and Corruption: Reflection on the State-Building Process*, Eur. J. of Sociology. 42(3), S. 27–71

Sally, D. (1995): *Conversation and Cooperation in Social Dilemmas: A Meta-Analysis of Experiments from 1958 to 1992*, Rationality and Society. 58(7)

Sen, A. K. (1974): *Choice, Orderings and Morality*, in Körner, S. (Hrsg.): Practical Reason: Papers and Discussions, Oxford, S. 54–67

Snidal, D. (1979): *Public Goods, Property Rights, and Political Organizations*, International Studies Quarterly. 23, S. 532–566

Somaya, D. (2003): *Strategic Determinants of Decisions not to Settle Patent Litigation*, Strategic Management J. 24(1), S. 17–38

Stallman, R. (1999): *The GNU Operating System and the Free Software Movement*, in: Dibona, C. /. Ockman, S. / Stone, M. (Hrsg.): Open Sources: Voices from the Open Source Revolution, Sebastopol, CA, S. 53–70

Torvalds, L. (1998): *FM interview with Linus Torvalds: What Motivates Free Software Developers*, Firstmonday. 3(3)

Tuomi, I. (2000): *Internet, Innovation, and Open Source: Actors in the Network*, Firstmonday. 6(1)

Ullman, E. (1997): *Close to the Machine: Technophilia and its Discontents*, New York, NY

Von Hippel, E. (1988): *The Sources of Innovation*, New York, NY

Von Hippel, E. (2001): *Innovation by User Communities: Learning from Open Source Software*, Sloan Management Rev. 42(4), S. 82–86

Von Krogh, G. / Spaeth, S. / Lakhani, K. (2003): *Community, Joining and Specialization in Open Source Software Innovation: A Case Study*, Res. Policy. 32(7), S. 1217–1241

Weber, M. (1973): *Die Objektivität sozialwissenschaftlicher und sozialpolitischer Erkenntnis*, in: Winckelmann, J. (Hrsg.): Gesammelte Aufsätze zur Wissenschaftslehre. Tübingen, S. 146–214

Wellman, B. / Gulia, M. (1999): *Virtual Communities as Communities*, in: Smith, M. A. / Kollock, P (Hrsg.): Communities in Cyberspace. New York, NY, S. 167–194.

Open Source-Geschäftsmodelle

RAPHAEL LEITERITZ

Einleitung

Dieser Text beschäftigt sich mit Open-Source-Software (OSS) auf Anbieterseite, also mit Unternehmen, die OSS als Grundlage ihres Geschäftsmodells haben. Er gibt Antworten auf u.a. die folgenden Fragen: Welche Möglichkeiten gibt es, mit OSS Geld zu verdienen? Kann ein Unternehmen mit etwas, das kein Geld kostet, profitabel arbeiten? Wie unterscheiden sich OSS-Geschäftsmodelle, welche OSS-Geschäftsmodelle sind tragfähig? Wie haben sich OSS-Geschäftsmodelle verändert und weiterentwickelt?

Grundlage dieses Textes ist eine Diplomarbeit, die der Autor 2002 an der Technischen Universität Berlin geschrieben hat. Der Autor ist darüber hinaus seit Mitte der neunziger Jahre Kenner der Open-Source-Szene und hat von 1997 bis 2001 eines der damals größten europäischen Linux-Unternehmen geleitet.

Nachfrager, also die Kunden, die sich für OSS und damit auch für Anbieter entscheiden, und deren Motive und die damit einhergehenden Eigenschaften von OSS werden an dieser Stelle nicht behandelt, hier verweist der Autor auf seine Homepage[1] und auf eine Reihe von weiteren aktuellen Quellen[2]. Generell kann man aber festhalten, dass sich OSS und insbesondere Linux in den letzten Jahren u.a. auf Grund der hohen Sicherheit und Stabilität, der Herstellerunabhängigkeit, der Offenheit/Flexibilität und des Preises bei Nachfragern in bestimmten Bereichen durchgesetzt hat. Als Produktsegment sind hier besonders der Server- und der Internet-Bereich zu nennen, als Marktsegment die öffentliche Verwaltung[3], der Mittelstand und Abteilungen in großen Unternehmen.

Außerdem kann an dieser Stelle aus Platzgründen nicht auf die Auswirkungen von Linux auf Microsoft und die Gegenstrategie eingegangen werden, auch hier sei auf aktuelle Quellen verwiesen.[4]

---

1 Http://www.leiteritz.com.

2 „Studie: Deutscher Open-Source-Markt gedeiht“ http://www.heise.de/newsticker/data/ola-02.07.03-001/ (21.7.2003), „Marktforscher: Linux legt in Europa zu“ http://www.heise.de/newsticker/data/anw-17.06.03-002/ (21.7.2003), „Merrill Lynch spart dank Linux“ http://www.heise.de/newsticker/data/ola-08.06.03-003/ (21.7.2003), „Studie: Open Source entlastet die Firmenkasse“ http://www.heise.de/newsticker/data/ola-07.05.03-005/ (21.7.2003)

3 „Innenministerium stellt Software-Migrationsleitfaden vor“, http://www.heise.de/newsticker/data/ola-10.07.03-004/ (21.7.2003), „Schily: Software-Vielfalt statt Monopolkultur“ http://www.heise.de/newsticker/data/jk-23.06.03-003/ (21.7.2003)

4 „Microsoft Deutschland bestellt ‚Behörden-Beauftragten‘“, http://www.heise.de/newsticker/data/jk-18.07.03-006/ (21.7.2003), „Microsoft will in Europa gegen Linux Boden gutmachen“, http://www.heise.de/newsticker/data/ola-02.07.03-002/ (21.7.2003), „Bill Gates: Keine Software für Linux“ http://www.heise.de/newsticker/data/ola-30.06.03-003 (21.7.2003), „Steve Ballmer fühlt sich durch Linux bedroht“ http://www.heise.de/newsticker/data/ola-05.06.03-001/ (21.7.2003), „Microsoft kämpft mit Sonderfonds gegen Linux“ http://www.heise.de/newsticker/data/ola-

Ebenso wird in diesem Text davon ausgegangen, dass die Grundlagen und die Geschichte von OSS bekannt sind. An dieser Stelle soll nur eine Begriffsklärung und Abgrenzung erfolgen, die für das Verständnis der späteren Ausführungen hilfreich ist: Unter OSS versteht man Software, deren Quellcode („source code"), also die geschriebenen Anweisungen des Programmierers, im Gegensatz zu herkömmlicher Software („proprietärer" oder „closed source"-Software) frei zugänglich ist. Die wesentlichen Eigenschaften von OSS sind:[5]

- Jeder hat das Recht, die Software nach eigenem Ermessen zu nutzen.
- Der Quelltext muss jedem Benutzer offen gelegt werden, oder es muss auf eine frei zugängliche Stelle verwiesen werden, wo er erhältlich ist.
- Der Benutzer hat das Recht, die Software zu modifizieren und in modifizierter Form weiterzuverteilen.
- Die Lizenz darf niemanden beim Verkauf oder bei der Weitergabe der Software in Form einer Softwarezusammenstellung einschränken.

Im Gegensatz hierzu gilt für proprietäre Software:

- Vervielfältigung, Weiterverbreitung und Modifizierung sind untersagt.
- Das Nutzungsrecht wird in Form einer Lizenz erteilt.
- Eigentümer der Software ist nicht der Anwender, sondern weiterhin der Hersteller, denn er hat das Urheberrecht und die vollständige Kontrolle über das Produkt.

Mit der Aussage über die Quellcodeverfügbarkeit wird keine Aussage darüber gemacht, ob eine Software kostenpflichtig ist. OSS kann sowohl kostenfrei als auch entgeltpflichtig sein. Das Gleiche gilt auch für proprietäre Software (die z.B. in Form von Freeware zwar proprietär, aber kostenfrei ist)[6]. Im Folgenden werden die beiden Begriffe „OSS" und „proprietär" als Beschreibung für Software mit offenem und verborgenem Quellcode benutzt. Wenn zusätzlich eine Aussage über den Kostenaspekt der Software getroffen weden soll, wird der Begriff „kommerziell" verwendet.

2. Wertschöpfungskette

Dieser Abschnitt beschäftigt sich mit den allgemeinen Grundlagen von Geschäftsmodellen im IT-Bereich und den spezifischen OSS-Geschäftsmodellen und deren Vergleich.

Die Prozesse in einem Unternehmen können anhand einer Wertschöpfungskette gegliedert werden. Wie sehen die Schritte der Wertschöpfungskette aus Sicht von IT- Anbietern genau aus?

15.05.03-002/ (21.7.2003), „Microsofts Leitfaden gegen Linux" http://www.heise.de/newsticker/data/anw-08.04.03-006/ (21.07.2003), „Microsoft: Linux ist Risiko Nummer 2 für unser Geschäft" http://www.heise.de/newsticker/data/hos-20.07.03-001/ (21.7.2003)

5 Für eine vollständige Definition von OSS vgl. www.opensource.org.

6 Eine umfassende Abgrenzung von Software-Kategorien ist unter www.gnu.org/philosophy/categories.de.html zu finden.

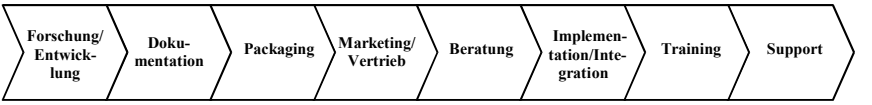


Abbildung 1: Allgemeine Software Value Chain (angelehnt an Cimetiere o.J. und Zerdick u. a. 1999)

Forschung und Entwicklung ist der Abschnitt der Wertschöpfungskette, in dem Software erstellt wird. Bei der *Dokumentation* wird Information zur Software und zum Einsatz der Software angefertigt.[7] *Packaging* bezeichnet den Vorgang, in dem die Einzelteile eines Produktes (Software, Dokumentation usw.) in einem Paket zusammenfasst werden. Bei kommerzieller Software ist dies üblicherweise ein Regalprodukt („commercial off the shelf"), bei OSS kann dies ein Softwarepaket sein, das im Internet bereitgestellt wird.

Im Bereich *Marketing/Vertrieb* sind die Vermarktungs- und Absatzaktivitäten eines Unternehmens zusammengefasst. Im Marketing definiert das Unternehmen, welche Produkte wie an wen verkauft werden. *Beratung* beschreibt die Unterstützung des Kunden vor der eigentlichen Softwareimplementation. Dazu gehören das Anfertigen von Studien, Analysen und Konzepten und gegebenenfalls die Anpassung der Unternehmensprozesse an die implementierte Software. *Implementation/Integration* ist der Abschnitt, in dem die Software vor Ort installiert wird.

Training beschreibt die Schulung des Kunden durch den Dienstleister. Im Bereich *Support* werden die installierten IT-Systeme im laufenden Betrieb betreut.

3. Produkt-Geschäftsmodelle

Aufbauend auf der allgemein definierten Wertschöpfungskette können wir nun die speziellen Geschäftsmodelle und deren Unterschiede betrachten. Wir beginnen mit den Produkt-Geschäftsmodellen in Abgrenzung zu den Dienstleistungs- und Mediator-Geschäftsmodellen.

3.1. Geschäftsmodell OSS-Distributor

OSS-Distributionen fassen OSS-Komponenten auf Datenträgern zusammen und machen sie durch Installations- und Administrationsroutinen als Komplettprodukt nutzbar. Das Geschäftsmodell der Distributoren besteht aus Entwicklung, Vermarktung, Vertrieb und Support dieser Distributionen.

Im Folgenden werden Linux-Distributionen als Beispiel für OSS-Distributionen untersucht (andere OSS-Distributionen sind z.B. OpenBSD und FreeBSD). Die wichtigsten Linux-Distributoren sind SuSE (Deutschland), Red Hat (USA), Turbolinux (USA/Japan), Caldera (USA) und Mandrake (Frankreich, Schwerpunkt auf Desktop-Linux).

Der überwiegende Teil einer Linux-Distribution besteht aus frei verfügbaren OSS-Komponenten. Rosenberg (2000) schätzt beispielsweise, dass 87% einer Red Hat-Linux-Distribution nicht von Red Hat stammen. Nur wenige Komponenten,

[7] Bei OSS ist es üblich, den Quellcode ausführlich zu dokumentieren, sodass er auch von Dritten nachvollzogen werden kann.

wie z.B. Installationsroutinen, Administrationsoberflächen oder die Hardwareerkennung, werden selbst entwickelt. Manche Linux-Distributoren geben diese Kernkomponenten nicht als OSS frei,[8] sondern bieten sie nur als proprietäre Software an. In diesem Fall unterliegt die Distribution nicht mehr vollständig der General Public Lincence (GPL) und darf sie nicht mehr ohne Einschränkungen kopiert werden.

Zusätzlich zu eigenen Entwicklungen werden von den Distributoren auch fremde OSS-Projekte gefördert. So fördert z.B. Red Hat das Entwicklungsprojekt „GNOME“[9]. Diese unabhängigen Entwicklungen kommen allerdings nicht nur Red Hat, sondern auch Mitbewerbern zugute.

Zusätzlich zum Produkt wird ein Hersteller-Support (Telefon, E-Mail) angeboten, der für einen bestimmten Zeitraum kostenlos ist. Das Angebot wird in der Regel auf einen definierten Umfang (Installationsprobleme) begrenzt. Darüber hinaus bieten manche Distributoren eine Wissensdatenbank mit Problemlösungen an, die kostenfrei bzw. durch Registrierung genutzt werden kann.

Alle darüber hinaus gehenden Leistungen werden als professionelle Dienstleistungen (im Kapitel 4 als „OSS-Dienstleister“ beschrieben) angeboten.

3.1.1. Marktpositionierung

Die Distributoren bewegen sich im Markt der Computer-Betriebssysteme für Server, Clients, und Embedded Systems. Zielgruppen der Distributoren sind sowohl Unternehmen (dort die IT-Entscheider und -Administratoren) als auch Privatkunden.

Distributionen werden in Releases zusammengefasst und mit einer Versionsnummer versehen. Sie werden dabei nach verschiedenen Versionen (z.B. „Red Hat 7.2“), Hardwareplattformen (z.B. Intel, Itanium, Embedded) und Zielkunden (Red Hat Linux und Red Hat Linux Professional) unterschieden. Releases werden im Abstand von wenigen (bei Red Hat und SuSE: drei) Monaten produziert und auf Datenträgern wie CD und DVD gepresst und/oder auf Internetservern abgelegt.

Viele Distributoren bieten ein Standardprodukt für Endkunden und eine Professional-Version für Unternehmenskunden an. Die Professional-Version unterscheidet sich dabei technisch nur wenig vom Standardprodukt, ist aber mit zusätzlichen Angeboten auf den Unternehmensmarkt zugeschnitten (24h-Telefonhotline, Schulungsangebot). Teilweise wird auch kommerzielle und/oder proprietäre Software mitgeliefert.

OSS-Distributoren verfolgen eine Branding-Strategie (Aufbau einer hochwertigen Marke). Bob Young, Gründer von Red Hat, bezeichnet die Red Hat Distribution als „commodity“ und vergleicht die Strategie seines Unternehmens mit der eines Ketchupherstellers (Dibona 1999):

„Ketchup is nothing more than flavored tomato paste. Something that looks and tastes a lot like Heinz Ketchup can be made in your kitchen sink without so much as bending a copyright rule. It is effectively all freely-redistributable objects: tomatoes, vinegar, salt, and spices“ (Dibona 1999, Kapitel „Giving it away“, Abschnitt „We Are in the Commodity Product Business“).

[8] Zum Beispiel die Installations- und Verwaltungssoftware „yast“ von SuSE Linux.

[9] Ziel dieses Projekts ist die Entwicklung einer einfachen und komfortablen Oberfläche.

Distributoren positionieren das Unternehmen gleichermaßen bei Endkunden, Unternehmenskunden und der Open-Source-Community. Die Linux-Distributoren setzen dabei auf einen Werbemix aus Produkt- (Zeitschriften und Internet) und Unternehmenswerbung (Messeauftritte und Imageanzeigen) und auf PR-Arbeit.

Die Linux-Distributoren verfügen in der Regel über mindestens drei Distributionskanäle. Erstens den Direktverkauf über die Internetseiten des Distributors, zweitens den Vertrieb über Presse- und Einzelhandelskanäle (Buchhändler, Zeitschriften etc.) und drittens Partnerschaften mit Computerherstellern. SuSE und Red Hat kooperieren z.B. mit IBM und Compaq (vorinstallierte Linux-Computersysteme) sowie kleinen und mittleren IT-Dienstleistern, die Distributionen im Rahmen von Projekten einsetzen. Die Partner können dabei auch Dienstleistungen von den Distributoren beziehen („1st level support", Wartungsverträge) und sind dann gleichzeitig auch Kunden.

Linux als prominentester OSS-Vertreter ist in der Vergangenheit vorrangig auf Serversystemen eingesetzt worden. In den letzten Jahren wird vermehrt auch der Einsatz auf dem Desktop diskutiert, beispielsweise im Zusammenhang mit der Umstellung des Münchner Stadtrats[10] auf Linux oder dem Migrationsleitfaden[11] des BMI. Einige Distributoren haben sich zu einem Desktop-Konsortium[12] zusammengeschlossen, andere konzentrieren sich sogar vollständig auf den Einsatz von Linux auf dem Desktop[13].

3.1.2. Gewinnmuster

Das Gewinnmodell der Distributoren besteht aus dem Verkauf von Linux-Distributionen auf Datenträgern über Partner an Endkunden. Dabei wird ein einmaliger Kaufpreis erzielt. Es fallen keine weiteren Gebühren oder Erlöse an. Die Distributoren bieten regelmäßig neue Releases an (bei SuSE ca. viermal pro Jahr). Manche Distributoren (z.B. Red Hat) legen die Distribution vollständig zum Download bereit. In diesem Fall wird mit dem Produkt überhaupt kein Umsatz erzielt. Außerdem bieten die Distributoren zusätzliche Dienstleistungen wie Support, Wartungsverträge und u.U. Integration/Implementation an.

3.1.3. Ressourcenfokus

Das Geschäftsmodell der Distributoren umfasst aus Sicht des Autors die Bereiche Forschung/Entwicklung, Dokumentation, Packaging, Marketing/Vertrieb und Support aus der Wertschöpfungskette.

---

10 „Münchens Clientstudie im Internet erhältlich" http://www.heise.de/newsticker/data/anw-27.06.03-003/default.shtml (21.7.2003).

11 „Innenministerium stellt Software-Migrationsleitfaden vor" http://www.heise.de/newsticker/data/ola-10.07.03-004/default.shtml (21.7.2003).

12 „Konsortium will Linux auf die Desktops bringen" http://www.heise.de/newsticker/data/anw-04.02.03-000/default.shtml (21.7.2003).

13 „LindowsOS 4.0 soll ungeahnt leicht zu bedienen sein" http://www.heise.de/newsticker/data/ola-25.06.03-003/ (21.07.2003).

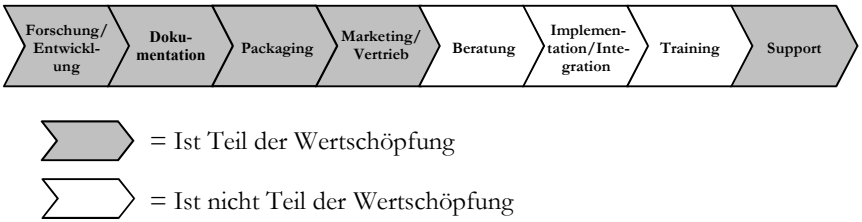


Abbildung 2: Ressourcenfokus der Distributoren

Partner der Distributoren sind z.B.

- Entwicklungspartner, die Software zuliefern (OSS-Entwickler oder Unternehmen)
- Marketing- und Vertriebspartner (Buchhändler, Hardwarehersteller)
- Dienstleistungspartner (z.B. große IT-Anbieter und regionale Systemhäuser, Dienstleister und Integratoren).

3.1.4. Strategische Absicherung

OSS-Distributoren haben im Gegensatz zu Herstellern proprietärer Software keine Möglichkeit, ihre Produkte zu schützen. Die Software auf den Distributionen steht unter der GPL und ist deshalb frei kopierbar. Teilweise dürfen sogar die gesamten Distributionen frei kopiert werden. Die Distributoren setzen deshalb zur strategischen Absicherung auf die Etablierung einer globalen Marke und eines Standards mit möglichst hoher Verbreitung. Die Markteintrittsbarrieren für Mitbewerber sind dabei relativ gering, denn die OSS-Komponenten der Produkte sind für jeden Mitbewerber frei verfügbar.

Strebt ein Kunde einen Systemwechsel auf eine andere Betriebssystemtechnologie (Windows, Unix oder z.B. SuSE zu Red Hat) an, entstehen Umstellungskosten. Dadurch werden die Kunden an die eigene Distribution gebunden.

Da die Preise für die Linux-Distributionen niedrig sind – im Vergleich mit kommerziellen Produkten sogar sehr niedrig – kann man von einer relativ hohen Preissensitivität der Kunden ausgehen, insbesondere im Privatbereich. Kommerzielle Softwareanbieter können proprietäre Produkte deshalb nicht über Preisstrategien, sondern nur über Leistungsdifferenzierung in den Markt einführen.

Mögliche Bedrohungen für die Produkte der Distributoren sind alternative Bezugswege für die Distribution. Bedeutsam ist hierbei vor allem der Download der Distribution aus dem Internet, da hier kein Umsatz generiert wird. Im Moment kann diese, von den Distributoren sogar geförderte, Möglichkeit mit der Marken- und Standardisierungsstrategie begründet werden. Langfristig nimmt diese Bedrohung zu, da immer mehr Anwender über Hochgeschwindigkeitszugänge verfügen und das Internet als Vertriebskanal an Bedeutung gewinnt.

Eine weitere Bedrohung der Distributionen sind vorinstallierte Computersysteme. Bei wachsendem Marktanteil ist davon auszugehen, dass die Zahl der vorinstallierten OSS/Linux-Computersysteme zunehmen wird. Wenn die Distribution kom-

plett unter der GPL steht – wie im Falle von Red Hat – muss der Computerhersteller dabei keine Lizenzabgaben oder sonstige Abgaben an den Distributor zahlen, der Umsatz für den Distributor fällt also in kompletter Höhe weg. SuSE hat kürzlich eine Partnerschaft mit WalMart gestartet und bietet vorinstallierte Rechnersysteme in den USA an.[14]

Eine theoretische Bedrohung der Marktstellung der Distributoren liegt im Verhältnis zu den Entwicklern. Sollte aus irgendeinem Grund – beispielsweise durch die Verletzung der GPL oder den „Missbrauch" einer Marktposition – ein Linux-Distributor den Unmut der Open-Source-Community auf sich ziehen, wäre ein Boykottaufruf der OSS-Entwickler denkbar. Es ist allerdings kein solcher Fall bekannt.[15] Selbst wenn es dazu käme, muss das wechselseitige Abhängigkeitsverhältnis der Distributoren und Entwickler berücksichtigt werden. Obwohl die Entwickler kein Geld bekommen, sind sie dennoch auf die Verbreitung der Software angewiesen, denn sonst macht die Entwicklung nur begrenzt Sinn.

3.1.5. Organisation, Mitarbeiter und Kultur

Im Rahmen der Branding-Strategie bauen die Distributoren eine internationale Vor-Ort-Präsenz auf. Dabei ist Red Hat, der erste profitable Dienstleister[16], am weitesten fortgeschritten. SuSE und Mandrake (allerdings wirtschaftlich stark geschwächt[17]) haben einen europäischen, Turbolinux hat einen asiatischen Schwerpunkt. Die drei letztgenannten haben sich 2003 zu der UnitedLinux-Initiative zusammengeschlossen und bündeln Entwicklungs-, Vermarktungs- und Vertriebskräfte.[18]

Die Organisation der Distributoren teilt sich zum Einen in zentrale Softwareentwicklung und zum Anderen in Dienstleistungen, die an den jeweiligen Unternehmensstandorten angeboten werden. Für die Softwarentwicklung benötigen die Distributoren qualifiziertes Entwicklungspersonal. Für den Dienstleistungsbereich werden u. a. Berater, Hotline-Mitarbeiter und Systemadministratoren benötigt.

Einige Distributoren unterstützen ihre Branding-Strategie dadurch, dass sie prominente OSS-Entwickler als Mitarbeiter verpflichten. Dadurch entsteht aus Sicht der Kunden eine Qualitätsaufwertung, denn die Technologie wird „aus erster Hand" geliefert.

---

14 „SuSE-Rechner bei Wal-Mart.com" http://www.heise.de/newsticker/data/anw-15.07.03-009/ (21. 07. 2003).

15 Ein deutscher Physikstudent ruft im Internet dazu auf, mit Hilfe einer Vereinskonstruktion 25% der Aktien von SuSE zu übernehmen, weil dem Autor „sowohl das Konzept als auch die Umsetzung der Firma SuSE GmbH AG in Bezug auf die Umsetzung des freien Betriebssystem Linux nicht paßt" (www.physik.tu-cottbus.de/users/heinold/suse/). Das Projekt klingt nicht nur abenteuerlich, sondern ist auch seit Anfang 2000 nicht aktualisiert worden.

16 „Red Hat weiter im Aufwind" http://www.heise.de/newsticker/data/ola-18.06.03000/default.shtml (22. 6. 2003).

17 „Linux-Distributor MandrakeSoft will aus Gläubigerschutz heraus", http://www.heise.de/newsticker/data/see-02.07.03-000/default.shtml (22. 6. 2003).

18 Vgl. http://www.unitedlinux.com.

3.2. Geschäftsmodell OSS-Applikationsanbieter

Das vorige Geschäftsmodell hat sich mit den Distributionen beschäftigt. Die Wertschöpfung der Distributoren besteht vor allem darin, fremde Software in Paketen zusammenzustellen und zu vertreiben. Dieses Kapitel beschäftigt sich nun mit dem Fall, dass ein Unternehmen eigene Software entwickelt und unter eine OSS-Lizenz stellt.

Bei der Betrachtung von OSS-Applikationsanbietern können drei Fälle unterschieden werden:

- Fall 1: Ein Unternehmen gibt eine Software, die es zu einem früheren Zeitpunkt proprietär entwickelt hat, ab einem bestimmten Zeitpunkt im Quellcode frei. Hier kehrt das Unternehmen den klassischen OSS-Entwicklungsprozess um und „konfrontiert" es die OSS-Welt mit einem fertigen OSS-Produkt (Beispiel: Netscape mit dem Netscape-Browser, genannt Mozilla).
- Fall 2: Ein Unternehmen beginnt, eine Software ab einem bestimmten Zeitpunkt unter einer OSS-Lizenz zu entwickeln. Dieser Fall verläuft analog zum klassischen OSS-Entwicklungsprozess. Der einzige Unterschied besteht darin, dass keine bzw. nur wenige unabhängige Personen an dem Projekt arbeiten, da ein einzelnes Unternehmen den Prozess dominiert (Beispiel: Red Hat/GNOME).
- Fall 3: Ein Unternehmen „übernimmt" zu einem bestimmten Zeitpunkt ein bis dato existierendes OSS-Projekt und betreut dieses ab diesem Zeitpunkt kommerziell. Hier wechselt der Charakter des OSS-Modells zu diesem Zeitpunkt von frei auf kommerziell.[19]

3.2.1. Marktpositionierung

Die Anbieter dieses Geschäftsmodells bewegen sich im gesamten Markt für Software-Applikationen. Dieser umfasst alle Arten von Betriebssystemen, Hardwareplattformen und Anwendungszwecken.

Für die Betrachtung des Geschäftsmodells ist es egal, um welche Art von Software es sich genau handelt. Es könnte sich z.B. um eine kleine Zusatzsoftware handeln oder um eine komplette Office-Suite. Der zu Grunde liegende Mechanismus des Geschäftsmodells ist davon nicht betroffen. Je nach Art der Software variiert entsprechend auch die Zielgruppe.

Die Vermarktung der Software findet meist mit Hilfe der OSS-Community statt. Je „glaubwürdiger" und akzeptierter das Gewinnmodell ist, desto mehr wird die Community die Software bekannt machen und nutzen. Dies geschieht z.B. durch Webseiten, in Newsgroups, auf Messen und in Zeitschriften. Die Software wird in der Regel auf Projekt-Homepages im Internet oder bei OSS-Mediatoren (siehe Kapitel 5) bereitgestellt.

Der Vetrieb der Produkte erfolgt über das Internet, in seltenen Fällen können Datenträger und Handbücher bestellt werden (z.B. bei StarOffice).

[19] Gründe für diesen Fall können z.B. sein, dass der oder die Hauptentwickler das Projekt nicht mehr weiterbetreuen können oder wollen, und dass das Projekt so erfolgreich ist, dass es zu viele Ressourcen beansprucht.

3.2.2. Gewinnmuster

Beim Geschäftsmodell OSS-Applikationsanbieter können folgende Gewinnmuster unterschieden werden:

Verschenken der Software [20]

Hier gibt der Anbieter die Software komplett unter einer OSS-Lizenz frei. Dem Anbieter ist es praktisch nicht mehr möglich, mit dem Verkauf der Software Erlöse zu erzielen (laut GPL zwar möglich, aber nicht praktikabel). Zwar behält der Anbieter oft noch eine Schlüsselfunktion (z.B. die Homepage des OSS-Projekts), aber in der Konsequenz befindet er sich auf der gleichen Stufe wie jeder andere Entwickler (und Mitbewerber!). Ein Beispiel für dieses Modell ist der Anbieter Sun mit dem OSS-Office-Produkt OpenOffice, das ursprünglich proprietär und kommerziell war.[21]

Diese Variante ist die radikalste Ausprägung des Geschäftsmodells. Der Anbieter verschenkt Software ohne primäre Gegenleistung und hat u.U. sogar noch laufende Kosten (Koordination des Entwicklungsprozesses, Infrastruktur, Personalkosten etc.). Für den Anbieter kann sich diese Variante wirtschaftlich nur lohnen, wenn er eine Überleitung zu Sekundärgeschäftsmodellen vornehmen kann.

Eine andere Motivation für das Verschenken der Software kann darin liegen, dass sich ein Anbieter von dem Produkt keinen kommerziellen Erfolg mehr verspricht. Dieser Schluss liegt z.B. beim Netscape-Browser/-Projekt Mozilla und beim Office-Paket Sun StarOffice/Projekt OpenOffice nahe: Beide Unternehmen haben den Wettbewerb gegen Microsoft verloren und machen „aus der Not eine Tugend". Anstatt einen bereits verlorenen Kampf gegen den Marktführer zu führen, attackieren sie ihn mit einer freien Variante. Mögliche Vorteile sind der Imagegewinn und die Schwächung der dominanten Position von Microsoft, die ja wiederholt zum Eintritt in neue Märkte genutzt wurde.

Lizenzierung nach Zeit

Dieses Modell stellt eine Variante des oberen Modells dar. Der wesentliche Unterschied ist, dass der Anbieter die Software in Versionen unterteilt und diese unterschiedlich behandelt. Aktuelle Versionen der Software sind kommerziell (nicht unbedingt proprietär). Ab einem bestimmten Zeitpunkt (feste Zeitspanne oder bei Erscheinen einer neuen Version) wird die Version unter einer OSS-Lizenz freigegeben.

Mit diesem Verfahren geht der Hersteller einen Mittelweg: Die Investitionen in die Entwicklung sind für einen Zeitraum geschützt und können wirtschaftlich genutzt werden.

Lizenzierung nach Zielgruppe

Diese Variante differenziert nicht nach Zeitpunkt, sondern nach Zielgruppe. Der Anbieter unterteilt die Kunden beispielsweise nach Privatkunden und Unternehmen (oder: kleine Unternehmen/große Unternehmen). Je nach Segment wird

[20] Für eine allgemeine Analyse (unabhängig von OSS), warum es für Unternehmen sinnvoll sein kann, Produkte zu verschenken, vgl. Busa (1999).

[21] In der Zwischenzeit wurde das Sun-eigene Produkt StarOffice, dem OpenOffice (Lizenz: LGPL) zu Grunde liegt, wieder kommerziell, es wird also nicht mehr kostenlos verteilt.

die Software dann kostenfrei oder kommerziell angeboten, in beiden Fällen ist sie OSS. Diese Variante ähnelt dem Shareware-Modell, denn auch dort wird teilweise nach Zielgruppe unterschieden (allerdings wird bei Shareware kein Quellcode mitgeliefert). Ein Beispiel für diese Variante war der deutsche E-Commerce-Anbieter IntraDAT (mittlerweile insolvent). Eine andere Zielgruppendifferenzierung findet bei der freien Datenbank MySQL (www.mysql.com) statt. Sie ist nur dann kostenpflichtig, wenn ein Dritter das Produkt verkauft oder in ein eigenes Produkt integriert.

Lizenzierung nach Leistungsumfang

In dieser Variante differenziert der Anbieter das Produkt in verschiedene Versionen, die einen unterschiedlichen Leistungsumfang besitzen. Eine Basiskomponente wird als OSS freigegeben. Wenn die Ansprüche des Benutzers wachsen, wird eine professionelle, kostenpflichtige Version angeboten. Dieses Modell operiert mit einem „Lockeffekt". Der Vorteil für den Anbieter ist die kostenlose Werbung, der Vorteil für den Anwender die Testmöglichkeit. Der dahinter liegende Mechanismus ist ebenfalls aus der Shareware-Welt bekannt.

Lizenzierung nach Zielplattform

Einige Anbieter lizenzieren Softwareprodukte nur auf bestimmten Plattformen als OSS. Beispielsweise ist die Softwarebibliothek QT des Unternehmens Trolltech kommerziell auf Windows-Systemen, aber kostenfrei für Unix- und Linux-Systeme erhältlich. Allerdings darf mit Produkten, die auf der freien Version basieren, kein Geld verdient werden.

Lizenzierung nach Komponenten

Eine weitere Variante ist die Offenlegung einer wirtschaftlich weniger bedeutsamen Komponente, von der der Hersteller strategisch profitiert. Der Anbieter SAP hat beispielsweise die Datenbank SAP DB als Komponente seines ERP-Systems freigegeben und erhofft sich von der Freigabe einen Innovationsschub und eine Schwächung des Mitbewerbers Oracle. Ein andere Form dieser Variante ist die Freigabe einer Client-Software, während das gleichzeitig notwendige Server-Produkt kommerziell und/oder proprietär bleibt.

Proprietäre Software für OSS

Einige Anbieter bewegen sich zwar im OSS-Markt, bieten aber selbst keine OSS-Software an. So verkauft z.B. der Anbieter Covalent proprietäre Zusatzsoftware zum Webserver Apache. Diese Geschäftsmodelle spielen in der weiteren Analyse keine Rolle, da es sich nicht um OSS-Geschäftsmodelle handelt. Im Grunde sind die Anbieter normale Softwareunternehmen, die kommerzielle, proprietäre Software für OSS-Betriebssysteme anbieten.

Überleitung zu sekundären Gewinnmodellen

Häufig beschränken sich die Anbieter von freier Software nicht auf das Softwaregeschäft, sondern bieten sie zusätzlich Sekundärleistungen an. Diese finden sich in den späteren Abschnitten der Software-Wertschöpfungskette (z.B. Hersteller-Support für die Software, Wartungsverträge, Dokumentation und Training). Rosen-

berg (2000) vergleicht diese Vorgehensweise mit dem Rasierproduktehersteller Gillette. Dessen Rasierer werden zu niedrigen Preisen angeboten bzw. sogar verschenkt (Primärprodukt). Der langfristige Ertrag wird mit dem wiederholten Kauf der Rasierklingen (Sekundärprodukt) erzielt. Aus Sicht des Autors hinkt dieser Vergleich allerdings etwas, da die Käufer der Gillette-Produkte auf Grund der proprietären Eigenschaften gezwungen sind, Gillette-Klingen zu verwenden (nur Original-Klingen passen). Genau solch ein systembedingter Zwang fehlt aber bei OSS! Ein Vertreter dieses Modells ist der Softwareanbieter Zope Corporations mit ihrem Applikationsserver Zope. Die Software ist Open Source, und der Hersteller bietet für sie ein umfangreiches Business-Service-Angebot.

3.2.3. Ressourcenfokus

Folgende Abbildung zeigt die Teilung in einen primären, nicht-kommerziellen Abschnitt und einen sekundären, kommerziellen Abschnitt der Wertschöpfungskette.

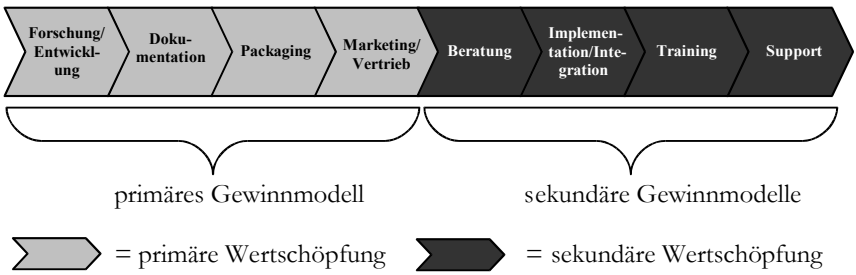


Abbildung 3: Wertschöpfungskette der OSS-Applikationsanbieter

Die OSS-Applikationsanbieter haben in der Regel keine Partner, da die Software direkt über das Internet vertrieben wird. Eventuelle Dienstleistungspartner würden mit dem sekundären Gewinnmodell konkurrieren.

3.2.4. Strategische Absicherung

Die strategische Absicherung der Anbieter erfolgt über zwei Faktoren: Preis und Standardsetzung. Der Preis ist ein starker Absicherungsfaktor, denn er kann in diesem Modell nicht unterboten werden. Für einige Mitbewerber wird es nicht möglich sein, die Software kostenlos freizugeben, z.B. weil der Anbieter zu klein ist oder weil die Software einen strategischen Faktor darstellt. Die Standardsetzung ist der zweite Absicherungsfaktor. Gelingt es einem Anbieter, mit einem OSS-Produkt einen Industriestandard zu setzen, schließt er Mitbewerber aus dem Wettbewerb aus. Im Sinne der Netzwerkeffekte verstärken sich einmal gesetzte Standards noch weiter, und wenn die Software noch dazu umsonst zu haben ist, werden die Kunden kaum zu Mitbewerbern wechseln.

Die Sekundärgewinnmodelle werden über das primäre Gewinnmodell abgesichert: Die Freigabe der Software belegt Kompetenz und Glaubwürdigkeit.

Die Käufer besitzen in der Regel wenig Macht, da sie wenig Druck aufbauen können, insbesondere wenn der Anbieter das einzige Unternehmen aus dem Marktsegment ist, der das Produkt verschenkt.

Mögliche Bedrohungen sind andere OSS- oder proprietäre Produkte. Für letztere bleibt nur die Leistungsdifferenzierung der eigenen Produkte z.B. durch bessere Qualität, mehr Leistungsumfang oder Support.

3.2.5. Organisation, Mitarbeiter und Kultur

Diese Faktoren unterscheiden sich nicht von den in der Softwarebranche üblichen Faktoren. Das Zentrum der Unternehmenstätigkeit ist die Softwareentwicklung. Es ist keine überregionale Organisation notwendig.

3.3. Geschäftsmodell OSS-Appliance-Hersteller

„Appliances" sind Geräte, die aus einer Hardware-Software-Betriebssystem-Kombination bestehen. Die Anbieter entwickeln zusätzlich zum OSS-Betriebssystemkern eigene Applikationen für die Schnittstelle zum Benutzer (Administration, Bedienoberfläche, Updatefunktionen und Ähnliches)[22]. Diese Applikationen sind meist proprietär, was aber nicht der GPL widerspricht, da die Software zwar zusammen aufgespielt ist, aber nicht logisch/technisch aufeinander aufbaut.

Der Support der Appliances wird in der Regel über Partner abgewickelt. Der Partner betreibt beispielsweise einen telefonischen Support und/oder eine Vor-Ort-Unterstützung (first level). Der Partner kann wiederum Support beim Hersteller einkaufen („second level" oder „third level"-Wartungsvertrag), wenn es um komplexere Probleme geht.

Beispiele für Appliance-Anbieter sind:

- Server-Appliances: Cobalt/Sun, IBM Whistle, VA Software, eSoft
- Thin Clients: IBM NetVista, Linware, Wyse
- Set top boxen: Nokia, PersonalTV, TiVo.

3.3.1. Marktpositionierung

Die meisten Appliance-Anbieter haben ein ganzes Portfolio von Appliances. Dieses gliedert sich nach Einsatzzweck (verschiedene Aufgaben) und nach Zielgruppe (Standard oder Professional). Zielgruppe der Appliances sind Unternehmen oder Privatpersonen, die die angebotenen Funktionen benötigen, diese aber nicht selbst installieren wollen oder können.

Appliances konkurrieren mit individuell konfigurierten Softwarelösungen. Bei diesen Lösungen fällt neben den Kosten für Hardware, Betriebssystem und Applikationen auch Arbeitsaufwand an. Deshalb sind Appliances in der Regel preiswerter. Manche Appliances werden dennoch in einem relativ hohen Preissegment angeboten. Diese zielen weniger auf den Massenmarkt, sondern werden auf Grund der

[22] Einige wenige Appliance-Anbieter entwickeln über die Benutzerschnittstelle hinaus auch basistechnische Komponenten (z.B. der deutsche Anbieter Astaro mit einer eigenen Firewall-Software für die angebotene Appliance).

Spezialisierung auf eine bestimmte Funktion als besonders hochwertige Lösung positioniert (Beispiel: Firewall-Appliances).

Appliances werden in der Regel über Partner vertrieben. Die Promotion-Aktivitäten werden deshalb primär im Umfeld der Partner durchgeführt (gemeinsame Messeauftritte und Anzeigenschaltung). Die Vermarktung der Produkte steht im Vordergund (im Gegensatz zur stärkeren Vermarktung des Herstellers wie z.B. bei den Distributor-Geschäftsmodellen).

Im Privatbereich werden Appliances über den Einzelhandel (Media Markt, Red Zac), im Unternehmensbereich über Partner (Distributoren, kleine IT-Systemhäuser, IT-Händler) vertrieben.

3.3.2. Gewinnmuster

Das Gewinnmodell besteht aus dem Verkauf der Appliance-Produkte über Partner an Endkunden. Hinzu kommen Erlöse aus Wartungsverträgen und Support-Anfragen der Partner und evtl. Updates[23]. Zusätzliche Erlöse können aus Partnerschaften mit Softwarelieferanten kommen, deren Produkte in die Appliances integriert werden (Vertriebsprovision).

3.3.3. Ressourcenfokus

Das Geschäftsmodelle der Appliance-Hersteller schließt aus Sicht des Autors die Teile Forschung/Entwicklung, Dokumentation, Packaging, Marketing/Vertrieb und Support ein:

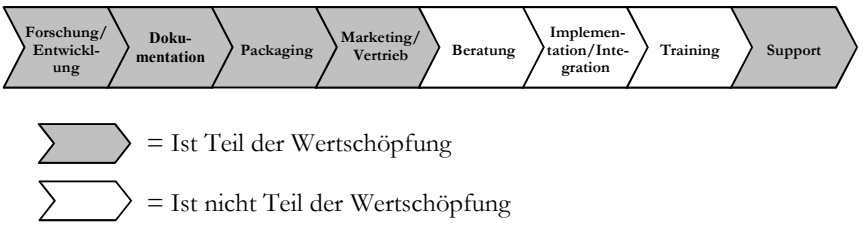


Abbildung 4: Wertschöpfungskette der Appliance-Anbieter

Partner der Appliance-Hersteller sind z.B.

- Entwicklungspartner, die Software zuliefern (OSS-Entwickler oder Unternehmen)
- Marketing- und Vertriebspartner (Großhändler, Hardwarehersteller, Einzelhändler)
- Dienstleistungspartner (kleinere Systemhäuser und Integratoren).

3.3.4. Strategische Absicherung

Die strategische Absicherung erfolgt über den Preis und die installierte Basis. Appliances sind Massenprodukte. Große Volumina sind entscheidend. Eine große

[23] Bei TV-Appliances (digitaler Videorekorder) existieren auch Subskriptionsmodelle, bei denen eine Art „intelligente digitale TV-Zeitschrift“ abonniert wird.

installierte Basis sichert wiederkehrende Umsätze über Updates, neue Appliances und Partner-Wartungsverträge.

Für Appliances gelten ähnliche Bedingungen wie für OSS-Distributionen. Der Appliance-Markt ist relativ preissensitiv, da es sich um einen Massenmarkt handelt. Bessere Einkaufskonditionen für die Hardware durch größere Volumina ermöglichen einen Marktvorteil. Ebenso entscheidend ist die Qualität der Vertriebskanäle. Je mehr (und bessere) Vertriebspartner ein Hersteller hat, desto besser sind seine Absatzschancen. Hier sind große Unternehmen mit bestehenden Handelsbeziehungen im Vorteil, da größerer Vertriebsdruck erzeugt werden kann.

Bedrohungen für OSS-Appliances sind:

- proprietäre Appliances (Windows-Appliances)
- individuell konfigurierte Softwarelösungen (internes Projekt mit OSS- oder proprietärer Software)
- Dienstleistungsangebote von Systemhäusern und Integratoren.

Auch Microsoft ist ein Wettbewerber, da mit der zunehmenden Integration von Komponenten in das Betriebssystem ebenfalls Funktionsbreite und Bedienkomfort erhöht werden. Ähnlich wie bei den Distributoren existiert das theoretische Risiko eines Boykotts durch die OSS-Community (Kapitel 3.1.4), es sind aber keine Fälle bekannt.

3.3.5. Organisation, Mitarbeiter und Kultur

Die Prozesse entsprechen auch hier den üblichen Softwareentwicklungsprozessen. Es ist keine globale Organisation notwendig, da der Vertrieb und die Vor-Ort-Installation durch Partner vorgenommen wird.

4. Dienstleistungs-Geschäftsmodell

Die bisher beschriebenen Geschäftsmodelle sind Produkt-Geschäftsmodelle. Bei ihnen liegt der Schwerpunkt auf der Entwicklung eines Softwareproduktes, der Freigabe unter einer OSS-Lizenz sowie dem Vertrieb und dem Vermarkten des Produktes.

Bei einem Dienstleistungs-Geschäftsmodell wird kein eigenes Produkt entwickelt, sondern es werden Dienstleistungen für existierende OSS-Produkte angeboten. Das OSS-Dienstleistungs-Geschäftsmodell hat sich zum „kleinsten gemeinsamen Nenner" der OSS-Geschäftsmodelle entwickelt. Fast alle Geschäftsmodelle rund um OSS haben (auch) einen Dienstleistungsanteil. Unterschiedlich ist vor allem die Angebotstiefe und -breite: Sie kann vom einfachen E-Mail-Support bis zur kompletten Dienstleistungspalette reichen.

Von allen OSS-Geschäftsmodellen ist das Dienstleistungsmodell am wenigsten umstritten, weil es herkömmlichen Geschäftsmodellen sehr ähnlich ist. Vergleicht man klassische IT-Dienstleister rund um proprietäre Software wie z.B. Accenture oder IBM Global Services mit OSS-Dienstleistern, sieht man faktisch wenige Unterschiede. In der Konsequenz ist es egal, ob man Beratung für betriebswirtschaftliche Standardsoftware oder OSS-basierte Firewalls anbietet. Die dahinter liegenden Prozesse und Mechanismen sind identisch.

Ein Unterschied zwischen proprietären und OSS-Dienstleistern besteht allerdings in der fehlenden Herstellerbeziehung. Viele proprietäre Dienstleister sind Partner von großen Softwareunternehmen. Diese binden die Dienstleister in Form von Partnerprogrammen an sich (Unterstützung in Form von Schulungen, Marketing und ermäßigten Lizenzen). Dies ermöglicht ein Zusatzgeschäft durch Softwarelizenzhandel. Ein typisches Beispiel für dieses Geschäft sind die Partner von großen ERP-Herstellern wie SAP, die eng mit dem Hersteller zusammenarbeiten und einen Teil ihres Umsatzes mit Lizenzhandel erzielen. Dieses Zusatzgeschäft fehlt OSS-Dienstleistern komplett. Natürlich könnten auch sie theoretisch nach dem o.g. Muster Partner von proprietären Softwareanbietern werden, allerdings verlassen sie damit den Markt der reinen OSS.

Die OSS-Dienstleister nehmen zusätzlich eine Qualitätssicherungsfunktion für die OSS-Komponenten ein. Dabei muss der Dienstleister testen, korrigieren und mit OSS-Entwicklern kommunizieren, was die Kosten zusätzlich erhöht.

IDA/Unisys (2000) kategorisiert die OSS-Dienstleister in vier Gruppen:

- Distributoren: Deren primäres Geschäftsmodell ist der Vetrieb einer OSS-Distribution. Zusätzlich bieten sie als sekundäres Geschäftsmodell auch Support für ihre Distribution an.
- Große Hardwarehersteller: Diese verkaufen Hardwaresysteme mit vorinstallierten OSS-Distributionen (hauptsächlich Linux), aber vertreiben keine eigene OSS-Distribution.
- Globale Systemintegratoren: Sie bieten weltweit IT-Dienstleistungen im proprietären, zunehmend aber auch im OSS-Bereich an.
- Spezialisierte OSS-Dienstleister: Diese Unternehmen konzentrieren sich vollständig auf Dienstleistungen für OSS-Komponenten.

4.1. Marktpositionierung

OSS-Dienstleister bewegen sich im Markt für IT-Dienstleistungen. Zielgruppe sind in der Regel IT-Entscheider in Unternehmen, keine Privatpersonen.

Dienstleister bieten kundennahe Wertschöpfungsschritte wie Beratung, Analyse, Konzeption, Integration, Netzwerkadministration, Sicherheitskonzepte, Implementation, Support, „remote management“ (Wartung von Systemen über Telefonleitung oder das Internet) und Training an.

IDA/Unisys (2000) unterteilt den Markt für Support-Produkte in vier Kategorien:

- Installations-Support: Die Distributoren gewähren beim Kauf einer Distribution für eine bestimmte Zeit kostenlosen Installations-Support.
- Support-Pakete: Distributoren bieten Support-Pakete in Form von „incident packs“ und „call packs“ an, in denen eine Anzahl von Vorfällen oder Anrufen zusammengefasst sind.
- Wartungsverträge: In diesen wird für eine gewisse Zeitspanne (3, 6, 9 oder 12 Monate) eine bestimmte Leistung durch den Anbieter garantiert.
- Integrationsleistungen: Allgemeine OSS-Dienstleister bieten über die einfache Hilfestellung hinaus auch komplexere Unterstützung für OSS an.

Die Vermarktung von Dienstleistungen unterscheidet sich von der von Produkten. Da Vertrauen und Kompetenz im Vordergrund stehen, sind Anzeigen weniger geeignet. Dienstleister setzen auf Kompetenzdarstellung im Rahmen von Konferenzen, Messen oder in Fachartikeln und „Mund-zu-Mund-Propaganda".

Dienstleister haben in der Regel einen Direktvertrieb mit direkter Kundenansprache durch Vertriebsmitarbeiter. Der Aktionsradius ist deswegen auf die Unternehmensstandorte begrenzt. Es es gibt keinen Multiplikatoreffekt durch andere Vertriebspartner (Ausnahme: Subunternehmeraufträge).

4.2. Gewinnmuster

Das Gewinnmuster im Dienstleistungsbereich besteht aus dem Verkauf von Personensätzen (überlichweise Stunden-, Tages- oder Monatssätze). Dabei ist der Auslastungsgrad der Mitarbeiter entscheidend, denn die Fixkosten (Personalkosten, Inftrastrukturkosten usw.) fallen unabhängig von der Auslastung an.

4.3. Ressourcenfokus

Aus Sicht des Autos können die Wertschöpfungsschritte der OSS-Dienstleister wie folgt beschrieben werden:

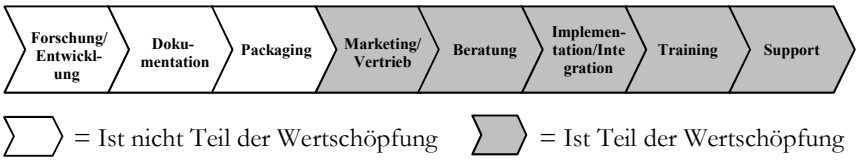


Abbildung 5: Der Ressourcenfokus der OSS-Dienstleister

Mögliche Partner der Dienstleister sind:

- Technologie-/Entwicklungspartner (Unternehmen bzw. OSS-Entwickler, die Technologie zuliefern)
- Dienstleistungspartner (Unternehmen, die der Dienstleister als Subunternehmer beschäftigt oder die den Dienstleister als Subunternehmer beschäftigen)
- Vertriebspartner (Unternehmen, die Dienstleistungen und Quasi-Produkte , wie Wartungsverträge, vermitteln)

4.4. Strategische Absicherung

OSS-Dienstleister befinden sich in einem stark fragmentierten Markt mit vielen Anbietern. Die Beziehung Kunde-Anbieter basiert auf Vertrauen und Kompetenz. Beides ist nicht schützbar und kann von jedem Anbieter aufgebaut werden. OSS-Dienstleister können versuchen, durch Beschäftigung und Unterstützung von prominenten Open-Source-Entwicklern oder die Förderung von OSS-Projekten eine besondere Kompetenz im OSS-Bereich aufzubauen, die sie von anderen Wettbewerbern unterscheidet.

Die Markeintrittsbarrieren im Dienstleistungsbereich sind sehr gering. Eindringlinge sind Dienstleister von proprietären Betriebssystemplattformen (Unix, Windows) auf der einen Seite und IT-Berater wie accenture auf der anderen Seite.

Wächst der OSS-Markt weiter, werden auch diese den OSS-Dienstleistungsmarkt betreten. Etablierte Dienstleister haben dabei den Vorteil, dass sie über weltweite Erfahrung und bestehende Kundenbeziehungen verfügen.

Die Kunden haben im Dienstleistungsbereich eine relativ große Macht. Im Gegensatz zum Produktgeschäft haben Dienstleister weniger Kunden mit *höheren* Umsätzen *pro* Kunde. Die Dienstleister gehen oft in Vorleistung, bevor Umsätze erzielt werden. Bei Streitigkeiten kann es sein, dass ein Kunde nicht bezahlen will (oder z. B. insolvent wird) und dadurch Mehraufwand oder Ausfälle entstehen.

Mögliche Bedrohungen für die OSS-Dienstleister sind die immer leistungsfähigeren Betriebssysteme und Softwareangebote. Dienstleistungen können nur verkauft werden, wenn der Kunde eine Leistung nicht selbst erbringen kann oder will. Wird Software immer einfacher bedienbar, sodass sie ohne Spezialwissen eingerichtet werden kann, sinkt der Bedarf an fremder Dienstleistung (Microsoft-Strategie).

Andere Bedrohungen sind Appliances (Kapitel 3.3), die Teile der Wertschöpfungskette in einfach zu bedienenden Komplettprodukten zusammenfassen und damit in Konkurrenz zu den Dienstleistungen treten. Theoretisch sind Boykottmaßnahmen der OSS-Entwickler denkbar, in der Praxis sind solche Fälle nicht bekannt.

4.5. Organisation, Mitarbeiter und Kultur

Die Organisation der OSS-Dienstleister ist auf Dienstleistungsprozesse für Kunden ausgerichtet (Beratung, Systemadministration, Projektcontrolling etc.). Die Kunden stehen im Vordergrund, deshalb unterscheidet sich die Kultur von Softwarentwicklungsunternehmen. Im Dienstleistungsbereich benötigen die Mitarbeiter neben Fachwissen auch „soft skills“.

Der Grad der Auslastung ist für die Profitabilität entscheidend. Deshalb müssen die Prozesse straff organisiert und durch geeignete Maßnahmen wie z.B. Zeiterfassung und Projektmanagement kalkulierbar, überprüfbar und abrechenbar sein.

5. Mediator-Geschäftsmodell

Mediatoren bringen verschiedene Interessengruppen im Umfeld von OSS über einen Marktplatz zusammen (Entwickler, Nutzer, Dienstleister, Werbetreibende).

Der bekannteste OSS-Mediator ist SourceForge, ein Tochterunternehmen des Appliance-Herstellers VA Software. SourceForge stellt eine technische Infrastruktur zur Verfügung, mit der Entwickler Software schreiben, ablegen, verwalten und kompilieren (in Maschinencode übersetzen) und die Entwickler miteinander kommunizieren können. Nutzer von OSS können über Projekt-Homepages Softwarepakete, Anleitungen und Dokumentationen downloaden.

5.1. Marktpositionierung

Die Zielgruppe der Mediatoren sind Entwickler, Nutzer, Dienstleister und Werbetreibende. Das Angebot für Entwickler besteht aus Verwaltung, Hosting und Vermarktung von OSS-Projekten. Das Angebot für Nutzer ist die zentrale Anlaufstelle für Recherche und Auswahl von OSS-Softwarekomponenten. Das Angebot für Sponsoren und Werbetreibende ist zielgruppenspezifische Werbung, das Angebot für OSS-Dienstleister sind die Vermarktung und der Vertrieb der Dienstleistungen.

Die Nutzung des Mediators ist für die Entwickler und die Nutzer kostenfrei. Sponsoren, Werbetreibende und Dienstleister zahlen teilweise für die Teilnahme am Marktplatz.

Marktplätze werden vor allem über das Internet vermarktet (Bannerschaltung und „Mund-zu-Mund-Propaganda").

5.2. Gewinnmuster

Das primäre Gewinnmodell der Mediatoren besteht darin, einen Marktplatz bzw. ein Portal zu betreiben und damit Erlöse zu erzielen.

Einnahmequellen der Mediatoren sind erstens der Verkauf von Bannern an Werbetreibende, zweitens Erlöse aus dem Vertrieb von Commerce-Produkten (Bücher, Datenträger) und drittens Einnahmen durch die OSS-Dienstleister. Subskriptionsmodelle in Form von kostenpflichtigen Abonnements sind theoretisch auch denkbar, auf Grund des hohen Anteils an fremdem Content von OSS-Entwicklern aber wohl nur schwer durchsetzbar.[24] Da kein Geld zwischen Nutzern und Entwicklern fließt, ist kein Provisionsmodell möglich.

5.3. Ressourcenfokus

Der Ressourcenschwerpunkt des Portalbetreibers liegt in Auswahl, Anpassung, Installation, Betrieb und Wartung des Marktplatzes. Bei den sekundären Geschäftsmodellen kommen Dienstleistungen und Softwareentwicklung hinzu.

5.4. Strategische Absicherung

Die strategische Absicherung der Anbieter erfolgt über den „Netzwerk- und Lock-In-Effekt"[25]: Ein erfolgreicher Marktplatz gewinnt auf Grund der hohen Anzahl an Teilnehmern zunehmende Attraktivität für neue Teilnehmer, wodurch der Marktplatz wiederum erfolgreicher wird (Bequemlichkeit, großes Angebot, Standardisierung). Dies wird wiederum eine steigende Benutzerzahl zur Folge haben. Die Wechselbereitschaft zu anderen Anbietern ist gering, da dort eine geringere Anzahl von OSS-Projekten und Nutzern zu finden ist. Anbieter können darüber hinaus keinen Preisangriff führen, da das Produkt ohnehin kostenfrei ist. Die Markeintrittsbarrieren für andere Mediatoren sind dadurch relativ hoch.

Die Nutzer des Portals haben eine relativ geringe Macht, da das Produkt für sie kostenfrei ist und deshalb kaum Druck auf den Anbieter aufgebaut werden kann.

---

[24] Es sei denn, die OSS-Entwickler würden finanziell davon profitieren, was aber vermutlich zu einem „Glaubenskampf" unter den OSS-Entwicklern führen würde.

[25] Siehe Zerdick 1999.

Bei den Distributor- und Appliance-Geschäftsmodellen wurden bereits mögliche Boykottmaßnahmen der OSS-Community beschrieben. Im Zusammenhang mit dem Mediatorenmodell gab es solche Aufrufe: Der Wechsel des Geschäftsmodells von SourceForge und die Kommerzialisierung und Proprietarisierung der dahinter liegenden Softwaretechnologie hat Unmut bei den OSS-Entwicklern erzeugt. Die Lobbyorganisation „Free Software Foundation", die insbesondere die Freiheitsaspekte der OSS vertritt, hat mit einem Boykottaufruf reagiert (Dachary 2001).

5.5. Organisation, Mitarbeiter und Kultur

Die Organisation des Unternehmens ist auf den Marktplatzbetrieb bzw. auf Softwareentwicklung ausgerichtet. Es sind keine globale Organisation und keine Vor-Ort-Dienstleistung durch Personen notwendig.

6. Sonstige Geschäftsmodelle

Der folgende Abschnitt beschäftigt sich mit sonstigen OSS-Geschäftsmodellen. Sie sind an dieser Stelle aufgeführt, weil sie nur am Rande mit OSS zu tun haben oder Kombinationen bzw. Variationen der oben beschriebenen Modelle darstellen.

6.1. Embedded OSS

Geschäftsmodelle für Embedded OSS sind im Wesentlichen Mischungen aus dem Distributions- und dem Dienstleistungsmodell und werden deshalb an dieser Stelle nur kurz skizziert. Die Distributionen sind speziell für den Einsatz auf kleinen Hardwareumgebungen (kleine CPU, wenig Speicher, Echtzeitumgebung) ausgelegt und optimiert. Cook (2000) beurteilt Linux aus folgenden Gründen positiv im Vergleich zu den typischerweise verwendeten „Eigenbau"-Betriebssytemen („home grown"): Linux ist standardisiert und Open Source. Es ist modular aufgebaut, kann auf spezifische Hardwareumgebungen angepasst werden, ist preiswert, und die Kompetenzen auf Entwicklerseite sind zahlreich vorhanden. Als nachteilig sieht er den vergleichsweise großen Umfang des Linux-Kernels, der relativ große Anforderungen an die Hardware stellt. Weiterhin bemängelt er die fehlenden Echtzeitfähigkeiten, Treiber und Hersteller-Support.

Die meisten Embedded-Distributionen werden unter OSS-Lizenzen vertrieben. Zusätzlich bieten die meisten Hersteller umfangreiche Services im Embedded-Bereich wie z.B. die Anpassung der Distribution auf spezielle Zielsysteme oder die Wartung von installierten Systemen an. Ein wichtiger Unterschied zu den OSS-Dienstleistungs- und -Distributions-Geschäftsmodellen besteht darin, dass die Kunden von Embedded-Unternehmen in der Regel Wiederverkäufer sind. Kunden können z.B. Hardwareprodukthersteller im Bereich Mobilfunk, Elektronik oder Automatisierungstechnik sein.

6.2. Vorinstallierte Hardware

Im Zuge des wachsenden OSS-Marktes haben einige Hardwarehersteller begonnen, ihre Hardwaresysteme auch mit Linux vorzuinstallieren (Beispiele: Dell, IBM und HP). Das Angebot besteht aus PC-Systemen (Server, selten Desktop) kombiniert mit üblichen Distributionen. Aus OSS-Sicht handelt es sich um ein typisches

PC-Hardware-Geschäftsmodell, es findet keine Wertschöpfung im OSS-Softwarebereich statt.

6.3. Große integrierte IT-Anbieter

Die großen IT-Anbieter wie z.B. IBM, Compaq, HP (aktuell mit einer Kooperation mit SuSE[26]) und Sun verfolgen integrierte OSS-Strategien. Die meisten von ihnen bieten OSS-Dienstleistungen (Schulung, Support, Beratung), OSS-Applikationen und vorinstallierte Hardware an. Auffallend ist die Tatsache, dass keiner der großen IT-Anbieter bisher eine eigene Distribution entwickelt oder einen Distributor übernommen hat. Vielleicht glauben sie, dass die OSS-Entwicklung ohne sie „natürlich" funktioniert und ein Eingreifen in den Markt Konsequenzen haben könnte (Boykott der Entwickler, Demotivation, neuer Kampf der Unix-Anbieter).

Der Anbieter IBM hat sich in vielerlei Hinsicht in den letzten Jahren als Open-Source-Förderer und -Nutznießer profiliert (Eigendarstellung: „Linux-Lokomotive"[27]). Bisher seien rund 1 Milliarde Dollar in Linux investiert worden.[28] Aus Sicht des Autors ist IBM in einer besonders guten Ausgangslage, um vom Linux- und Open-Source-Trend zu profitieren:

- IBM hat eine langjährige Unix-Tradition. Dabei ist Linux kein Substitut-, sondern vor allem ein Komplementprodukt zum konzerneigenen AIX, das technisch im absoluten High-End-Bereich positioniert ist. Eine „Kannibalisierung" von AIX durch Linux findet mutmaßlich nur vereinzelt statt.
- Für die Konzernsparten ist Linux ein perfektes Vertriebsargument. So kann sich beispielsweise die interne Dienstleistungssparte Global Services mit Linux-Kompetenz profilieren (Beratung, Training, Support etc.). Die Hardwaresparte wiederum kann eigene Server-Produkte speziell für Linux entwickeln.
- IBM hat wegen der Größe der Organsiation ganz eigene „economies of scale", um die Linux-Entwicklung zu fördern und daran zu partizipieren.
- Das Vertrauen in die Marke unterstützt IBM zusätzlich, in dem es z.B. eine eigene Sicherheitszertifizierung plant und dadurch einen Teil der Technologie nicht-technisch „proprietarisiert".[29]

Die Financial Times Deutschland urteilt ähnlich über die integrierten Anbieter, hier IBM und HP: Das „Randphänomen" Linux hat sich zu einem „zentralen Bestandteil der Geschäftsstrategie vieler Unternehmen" entwickelt.[30]

Sun tut sich mit einer Linux-Strategie dagegen etwas schwerer und hat kein so weit gehendes Engagement. Aus Sicht des Autors hat Sun große Bedenken, dass

---

[26] „Hewlett-Packard und Red Hat wollen enger zusammenarbeiten" http://www.heise.de/newsticker/data/anw-19.03.03-000/default.shtml (22.07.2003)

[27] „HP und IBM scheffeln Geld mit Linux" http://www.heise.de/newsticker/data/anw-23.01.03-002/ (21.7.2002).

[28] „HP und IBM scheffeln Geld mit Linux" http://www.heise.de/newsticker/data/anw-23.01.03-002/ (21.7.2002).

[29] „HP und IBM scheffeln Geld mit Linux" http://www.heise.de/newsticker/data/jk-14.02.03-010/default.shtml (21.7.2003).

[30] „HP und IBM scheffeln Geld mit Linux" http://www.heise.de/newsticker/data/anw-23.01.03-002/ (21.7.2002).

sich Linux zum Substitut für das hauseigene Solaris entwickelt. Deshalb wird von Sun auffällig stark darauf geachtet, Linux als „Einstiegssystem" zu positionieren und von Solaris abzugrenzen.[31]

Der Marktanteil und der Umsatz von Sun haben sich in den letzten Monaten verringert, was dafür spricht, dass Sun sich in einem strategischen Dilemma befindet: Solaris als einer der Hauptumsatzträger im High-End-Bereich wird von Linux attackiert, und die Einstiegsserver von Sun konkurrieren mit den immer erfolgreicheren Intel/Windows-Plattformen.[32]

Ein besonders auffälliges Beispiel für die Kannibalisierung eines Kernproduktes durch Linux zeigt die Strategie des Anbieters SCO. SCO ist viele Jahre lang der Marktführer für Unix-Systeme auf PC-Plattformen gewesen. Seit einigen Jahren macht SCO allerdings die direkte Konkurrenz von Linux schwer zu schaffen. Faktisch bietet Linux die gleiche, in vielen Punkten weitaus mehr Funktionalität als SCO Unix, sodass es kaum mehr Gründe gibt, das teure SCO-Produkt einzusetzen. SCO ist allerdings der Marken- und Patentinhaber von Unix, dem kommerziellen/proprietären Vorläufer von Linux.

Noch vor einigen Jahren sah es so aus, als würde SCO dem drohenden Bedeutungsverlust durch eine „Flucht-nach-vorn-Strategie" begegnen. So schlossen sich im Jahr 2000 SCO und der Linux-Distributor Caldera zusammen. Ziel des Mergers war die Integration der eigenen Unix-Kerntechnologie mit Linux zu einer „Open-Internet-Platform". Später gab SCO das Projekt in dieser Form auf und beteiligte sich an der UnitedLinux-Initiative der Red Hat-Wettbewerber SuSE, Mandrake und TurboLinux. Mitte 2003 ging SCO allerdings auf Konfrontationskurs: SCO wirft den Linux-Distributoren und IBM mittlerweile Verletzung des Patentschutzes vor, fordert eine Abgeltung durch eine „Linux-Steuer"[33] und plant evtl. sogar eine Klage gegen den Linux-Erfinder Linus Torvalds. Wie sich dieser Kampf entscheiden wird, ist unklar, aber SCO hat sich davon verabschiedet, durch eigene Produkte am Wachstum des Linux-Marktes zu partizipieren und bietet nun letzte Mittel auf, um das eigene Überleben zu sichern.

6.4. Embedded-Appliances

In der Unterhaltungselektronik- und Mobilfunkindustrie spielt OSS eine immer stärkere Rolle. So sind z.B. die ersten Mobiltelefone, PDAs und MP3-Spieler mit Linux-Betriebssystem auf dem Markt. Allerdings spielt OSS in dem Geschäftsmodell des Anbieters nur eine nachrangige Rolle. Der Käufer wird einen MP3-Player nach Kriterien wie Preis, Leistung, Bedienung und Größe auswählen und weniger,

---

[31] „Suns Blade-Server mit Athlon-Prozessoren kommen später" http://www.heise.de/newsticker/data/anw-13.07.03-007/ (21.7.2003), „SCO vs. IBM: Suns McNealy preist Vorzüge von Solaris" http://www.heise.de/newsticker/data/anw-18.06.03-002/ (21.7.2003), „Sun: Linux-Nutzer wollen in Wirklichkeit kein Linux" http://www.heise.de/newsticker/data/ola-12.06.03-003/ (21.7.2003), „Sun bringt Linux-PCs im Sommer" http://www.heise.de/newsticker/data/pmz-18.02.03-002/ (21.7.2003).

[32] „Suns Umsatz schrumpft weiter" http://www.heise.de/newsticker/data/jk-22.07.03-010/ (21.7.2003).

[33] „SCO vs. Linux: Linux-Steuer oder Sicherheit" http://www.heise.de/newsticker/data/jk-22.07.03-005/ (21.7.2003).

weil im Hintergrund Linux läuft. Allerdings kann OSS insofern durchaus bedeutend sein, dass auf die Entwicklung eines eigenen Betriebssystems und eigener Applikationen (bzw. die Lizenzierung durch einen Dritten) verzichtet und der Preisvorteil an die Kunden weitergeben werden kann. Sony äußert sich z.B. sehr positiv über Linux: Softwarebestände lassen sich bequemer als erwartet gemeinsam nutzen, außerdem könne der Kernel schnell angepasst und verbessert werden. Um Linux Heimelektronik-tauglich zu machen, bedürfe es der Mitarbeit einer großen Gemeinschaft von Entwicklern, genau die bringe Linux mit.[34] Motorola möchte gar langfristig alle Mobilfunkgeräte mit Linux ausstatten.[35]

6.5. Linux-Hotel

Ein Hotelanbieter in Essen vermarktet sein Hotel als voll ausgestattetes Linux-Hotel. Gäste können einen PC-Raum mit Linux-Rechnern benutzen. Motto des Anbieters: „Linux im Linuxhotel zu lernen, ist wie Englisch in England“[36].

6.6. Bücher, Gimmicks

Der Fachverlag O'Reilly gilt als *der* Fachverlag für Bücher zum Thema Open Source. Eine Reihe von bekannten Autoren haben dort hochwertige Bücher zu verschiedensten Aspekten von OSS veröffentlicht. Der Gründer Tim O'Reilly ist ein prominentes Mitglied der OSS-Community.

Eine Reihe von Anbietern haben den wachsenden OSS-Trend erkannt und bieten alle Arten von Artikeln und Gimmicks rund um OSS an. So gilt der Linux-Pinguin als ein beliebtes Plüschtiervorbild, genau wie sein (seltenerer) Kollege, der BSD-Teufel. Weitere Angebote sind diverse Tassen, Gebrauchsutensilien und T-Shirts rund um dem Themenkomplex OSS.

7. Zusammenfassung

Der folgende Abschnitt fasst zusammen, welche OSS-Geschäftsmodelle existieren, inwieweit sich die OSS-Geschäftsmodelle bewährt haben und welche Zukunft die OSS-Geschäftsmodelle haben.

Die OSS-Geschäftsmodelle unterliegen teilweise einer Transformation. Das folgende Schaubild zeigt die OSS-Geschäftsmodelle und deren Veränderungen.

[34] „Sony lobt Linux“ http://www.heise.de/newsticker/data/ola-30.05.03-004/default.shtml (21.7.2003).

[35] „Linux für Motorola-Handys“ http://www.heise.de//newsticker/data/rop-13.02.03-000/default.shtml (21.7.2003).

[36] Vgl. http://www.linuxhotel.de.

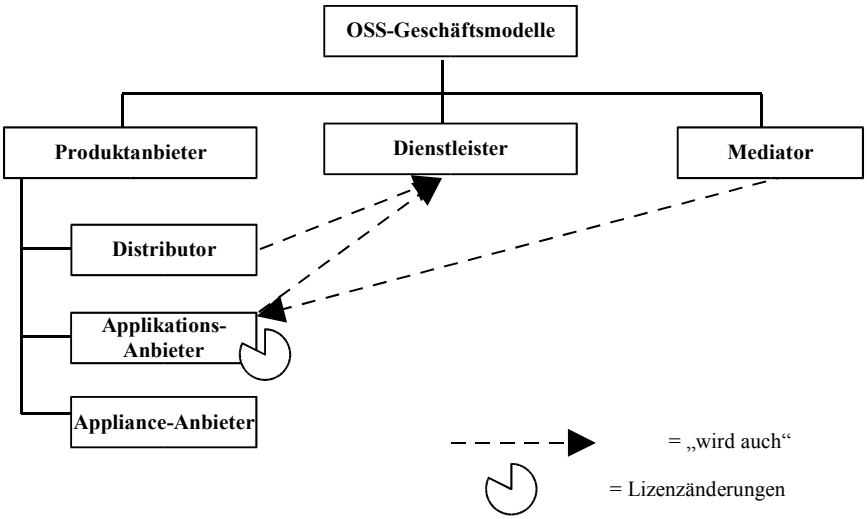


Abbildung 6: OSS-Geschäftsmodelle und deren Transformationen

Aus Sicht des Autors sind das Mediator-, das Distributions- und das OSS-Applikations-Geschäftsmodell als primäre Geschäftsmodelle nicht überlebensfähig. Nur in Verbindung mit sekundären Geschäftsmodellen (Dienstleistung/ kommerzielle Softwareangebote) können die Anbieter überleben.

Die OSS-Applikationsanbieter passen ihre Lizenzen auf Grund veränderter Marktbedingungen an. Teilweise werden OSS-Produkte rekommerzialisiert, einige Anbieter beendeten sogar ihre OSS-Aktivitäten komplett. Praktisch alle OSS-Applikationsanbieter haben Sekundärgeschäftsmodelle, bei denen sie indirekt aus der Freigabe der Software profitieren.

Die Distributoren entwickeln sich zunehmend (auch) zu OSS-Dienstleistern, weil der Verkauf der Distributionen offensichtlich nicht kostendeckend möglich ist.

Die Mediatoren haben ihr ursprüngliches Geschäftsmodell teilweise aufgegeben und verfolgen sekundäre Applikations-Geschäftsmodelle.

Das Appliance-Geschäftsmodell scheint aus Sicht des Autors prinzipiell überlebensfähig zu sein. Da es sich um Massenprodukte mit relativ niedrigen Margen handelt, müssen Anbieter große Stückzahlen absetzen. Bis heute ist es keinem Anbieter gelungen, signifikante Marktanteile mit OSS-basierten Produkten zu erreichen. Auch die Übernahme des Server-Appliance-Anbieters Cobalt durch Sun hat an dieser Tatsache offensichtlich nichts verändert (vgl. Shankland 2001b).

OSS-Dienstleistung hat sich zum „kleinsten gemeinsamen Nenner“ der OSS-Geschäftsmodelle entwickelt. Fast jeder Anbieter ist auch Dienstleister. Da sich dieses Geschäftsmodell nur in wenigen Aspekten von „normalen“ IT-Dienstleistungs-Geschäftsmodellen unterscheidet, ist dies nicht weiter überraschend. Allerdings hat das Dienstleistungsmodell eigene Herausforderungen (Auslastungsoptimierung, kein

„law of increasing returns"). Die herkömmlichen IT-Anbieter werden in den OSS-Dienstleistungsmarkt eindringen, und die OSS-Anbieter können sich auf Grund nicht existierender Schutzmechanismen nur mit Hilfe einer Markenstrategie verteidigen. Zusätzlich bleibt für OSS-Dienstleister das Problem, dass sie die Margen nicht durch Handel mit Softwarelizenzen aufbessern können und zusätzlich OSS-Qualitätssicherung übernehmen müssen.

Die großen integrierten IT-Anbieter, insbesondere IBM, kommen mit ganzer Macht auf den Markt, bedrängen spezialisierte OSS-Unternehmen und werden einen bedeutenden Teil des Umsatzes für sich gewinnen.

Literatur

Ardal, Atila (2001): *Open Source – das Beispiel Linux. Ökonomische Analyse und Entwicklungsmodell eines erfolgreichen Betriebssystems*, Diplomarbeit, online ig.cs.tu-berlin.de/da/059/ardal-opensource.pdf

Balzert, Helmut (1996): *Lehrbuch der Software-Technik: Software-Entwicklung*, Spektrum, Akademischer Verlag, Heidelberg

Balzert, Helmut (1998): *Lehrbuch der Software-Technik: Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung*, Spektrum, Akademischer Verlag, Heidelberg

Baumgartner, Jens D.: *Deutsche Übersetzung der „Open Source Definition"*, online www.wdrcc.de/sendungen/report/informationen/20000429/Thema01.htm

Bessen, James (2001): *OSS: Free Provision of Complex Public Goods, Research on Innovation*, Wallingford, online www.researchoninnovation.org/opensrc.pdf

Bezroukov, Nikolai (1999a): *A second look at the Cathedral and the Bazaar*, First Monday, Vol. 4, No. 12, December 1999, online www.firstmonday.dk/issues/issue4_12/bezroukov/index.html

Bezroukov, Nikolai (1999b): *OSS Development as a special type of academic research (a critique of vulgar Raymondism)*, First Monday, Vol. 4, No. 10, October 1999 online www.firstmonday.dk/issues/issue4_10/bezroukov/index.html

Blase, Paul (2000): *Open Source White Paper – A Software Company's Dilemma*, DiamondCluster, online www.diamondcluster.com/work/Wpapers/WPSoftware.asp

BMWi (2001): *Open-Source-Software-Ein Leitfaden für kleinere und mittlere Unternehmen*, online www.bmwi.de/Homepage/download/infogesellschaft/Open-Source-Software.pdf

Busa, Patricia (1999): *„Free" Pricing Model – Can businesses really make money by giving goods away for free?* University of Texas at Austin, online cci.bus.utexas.edu/research/white/free-price.htm

Claybrook, Bill (2001): *AberdeenGroup InSight: Linux is on the Move Up!* Aberdeen Research, July 24, 2001, online www.ibm.com/linux/LinuxInSight.pdf

Cimetiere, Jean-Christophe: *A New Era of Service Providers Driven by the Open Source Software (OSS) and Free Software wave*, Intranetjournal/TechMetrix Research, online www.intranetjournal.com/articles/200005/asp_05_30_00b.html

Cook, Rick: *Embedded Linux Basics*, LinuxWorld.com, May 2000
www.linuxworld.com/linuxworld/lw-2000-05/lw-05-embedded.html

Dachary, Loic (2001): *SourceForge drifting*, Free Software Foundation, online www.fsfeurope.org/news/article2001-10-20-01.en.html

Daffara, Carlo (2001): *Free Software / Open Source: Information Society Opportunities for Europe?* Working group on Libre Software, April 2001, online eu.conecta.it/paper/paper.html

Dafermos, George N. (2001): *Management and Virtual Decentralised Networks: The Linux Project*, First Monday, Vol. 6, No. 11, November 2001, online www.firstmonday.dk/issues/issue6_11/dafermos/index.html

Dibona, Chris / Ockman, Sam / Stone, Mark: *Open Sources. Voices from the Open Source Revolution*, O'Reilly, Sebastopol, CA, 1999, online www.sindominio.net/biblioweb/telematica/open-sources-html/main.html

Diedrich, Oliver (2000): *Schnellster Webserver unter Linux*, Heise online News, 4. Juli 2000, online www.heise.de/newsticker/data/odi-03.07.00-004/

Diedrich, Oliver (2001a): *Sprung auf den Desktop für Linux zu hoch?*. Heise online News, 15. Mai, online www.heise.de/newsticker/data/odi-25.05.01-002/

Diedrich, Oliver (2001b): *Linux-Kernel kann Dateisystem beschädigen*, Heise online News, 25. November, online www.heise.de/newsticker/data/db-25.11.01-000/

Diedrich, Oliver (2001c): *Sicherheitslücke im Linux-Kernel*, Heise online News, 19. Oktober, online www.heise.de/newsticker/data/odi-19.10.01-001/

Delio, Michelle (2000): *It'll Be an Open-Source World*, wired.com, August 15, 2000 online www.wired.com/news/print/0,1294,38240,00.html

Dyck, Timothy (2001): *MySQL*, CNET.com, March 5, 2001 online enterprise.cnet.com/enterprise/0-9513-707-6618957.html?tag=st.it.9500-704-0.sr.9513-717-6716336-7053536

Edlbauer, Florian (2001): *Studie: 75 Prozent lehnen Linux ab*, Zdnet.de-News, 24. März 2001, online news.zdnet.de/story/0,,s2056048,00.html

EITO (2001): *European Information Technology Observatory*

EDC (2001): *Embedded Systems Developer Survey 2001*, August 2001, online www.evansdata.com

Engdegard, Staffan (2001): *Jupiter MMXI: Preise für Online-Werbung sinken weiter*, Juli 2001, zitiert nach www.golem.de/0105/14072.html

Emnid (2001): *Bevölkerungsbefragung über Nutzung und Akzeptanz des Betriebssystems Linux*, eine repräsentative Befragung von TNS EMNID Telecommunication&IT im Auftrag der SuSE Linux AG, 24. März 2001

online www.suse.de/de/press/press_releases/archive01/misc/emnid_studie/index.html

Everitt, Paul (2001): *How we reached the Open Source Business Decision*, Digital Creations, online www.zope.org/Members/paul/BusinessDecision

Farbey, Barbara / Finkelstein, Anthony (2001): *Evaluation in Software Engineering: ROI, but more than ROI*, Paper submitted for „Third International Workshop on Economics-Driven Software Engineering Research" (EDSER-3 2001), online www.cs.virginia.edu/~sullivan/edser3/finkelstein.pdf

Forschungsstelle für Verwaltungsinformatik, Universität Koblenz-Landau (2000): *Protokoll der Veranstaltung „OSS (OSS) in der Bundesverwaltung" der Koordinierungs- und Beratungsstelle für Informationstechnik in der Bundesverwaltung, 26.–28. September*, online linux.kbst.bund.de/auftakt/ergebnis/oss-protokoll.pdf

Gallist, Rudi (2002): *Offener Brief an den Linux-Verband*, im Rahmen der Diskussion über die IT-Ausstattung des Deutschen Bundestags, Verband der Softwareindustrie Deutschlands,
online www.vsi.de/inhalte/spezial/spezial.asp?cid=Spezial&id=20

Galli, Peter (2001): *Linux takes a leap and a stumble*, Zdnet News, 20 Mai 2001,
online zdnet.com.com/2100-11-503728.html?legacy=zdnn

FSF/GNU: *Categories of Free and Non-Free Software*,
online www.gnu.org/philosophy/categories.html

Gartner Viewpoint (2001): *Linux support services: Like any other operating system?*,
online techupdate.zdnet.com/techupdate/stories/main/0,14179,2808791-1,00.html

Gehring, Robert A. (2001): *Software Patents – IT-Security at Stake?*, Informatik und Gesellschaft, TU Berlin,
online www.ig.cs.tu-berlin.de/ap/rg/2001-10/Gehring2001Full-SWPatITSec.pdf

Geipel, Philipp: *Intermediäre im Internet am Beispiel des B2C E-Commerce*, Seminar Marketing und Electronic Commerce, Institut für Entscheidungstheorie und Unternehmensforschung,
online www.corpssaxonia.de/seminararbeiten/bwl/Intermediaere_B2C_Vortrag.pdf

Gertz, Winfried (2001): *Das Consulting-Geschäft um Open Source*, Computerwoche, April 27, 2001,
online www2.computerwoche.de/index.cfm?pageid=267&type=ArtikelDetail&id=149632&cfid=2388386&cftoken=35343586&nr=8#

Ghosh, Rishab (1998): *Cooking pot markets: an economic model for the trade in free goods and services on the Internet*, First Monday, Vol. 3, No. 3, March 1998
online www.firstmonday.org/issues/issue3_3/ghosh/index.html

Ghosh, Rishab / Prakash, Vipul Ved (2000): *The Orbiten Free Software Survey*, First Monday, Vol. 5, No. 7, July 2000,
online www.firstmonday.dk/issues/issue5_7/ghosh/index.html

Gillen, Al / Kusnetzky, Dan / McLarnon, Scott (2001): *The Role of Linux in Reducing the Cost of Enterprise Computing. An IDC White Paper Sponsored by Red Hat Inc*,

IDC/Red Hat,
online www.redhat.com

Goder, Andrea (2001a): *Entdeckung der Linux-Welt*, Computerwoche Special, 02/2001

Goder, Andrea (2001b): *Pinguine und Pleitegeier: Open-Source-Firmen in der Konsolidierungsphase*, Computerwoche Special, 02/2001

Grassmuck, Volker (2000): *Geschichte und Mechanismen freier Software*, Wizards of OS-Kongreß,
online www.mikro.org/Events/OS/text/gesch-freie-sw.html

Hall, Mark (2000): *Start-up's Apps Make Desktop Linux Easier*, Computerworld, September 9, 2000,
online www.computerworld.com/cwi/story/0%2c1199%2cNAV47_STO63615%2c00.html

Hecker, Frank (1999): *Setting up Shop*,
online www.hecker.org/writings/setting-up-shop.html

Henkel, Thomas (2001): *Sooner or Later Even Linux Has to Turn a Profit*, Gartner Group,
online www.gartner.com/resources/96600/96692/96692.pdf

Hetze, Sebastian / Hohndel, Dirk / Kirch, Olaf / Müller, Martin: *Das Linux Anwenderhandbuch*, LunetIX GbR/ Softfair Verlag

Hillesley, Richard (2001): *Money to burn*, Linux User UK,
online www.linuxuser.co.uk/articles/issue10/lu10-Cover_feature-Money_to_burn.pdf

Hoch, Detlev J. (1999): *Secrets of Software Success: Management Insights from 100 Software Firms around the World*, Harvard Business School Press; Boston, Massachusetts

Hoffmann, Naomi (1999): *OSS*,
online public.kitware.com/VTK/pdf/oss.pdf

Howe, Carl D. (2000): *Open Source Cracks The Code*, Forrester Research,
online www.forrester.com/ER/Research/Report/Summary/0,1338,9851,FF.html

Ilan, Yaron (2001): *The Economics of Software Distribution over the Internet Revisited*, Firstmonday,
online www.firstmonday.dk/issues/issue6_12/ilan/index.html

IDC (2001): *Linux: What's the use? Western Europe 1999-2004*, IDC

Icaza, Miguel de (2001): *Is Linux ready for the corporate desktop?*, Computerworld, August 9, 2001,
online www.computerworld.com/storyba/0,4125,NAV47_STO62924,00.html

Jaeger, Till (2001): *Klage wegen GPL-Verletzung*, August 13, 2001,
online www.ifrOSS.de/

Jaeger, Till (2000): *Copyright oder Copyleft*, Computerwoche Spezial 4/2000, p. 36, 2000,
online www.ifrOSS.de/ifrOSS_html/art6.html

Jeong, Byung Seon (1999): *Analysis of the Linux Operating System, a New Entrant in the Operating System Markt: Technological Innovations and Business Model*, Dissertation

Johnson, Justin Pappas (2000): *Some Economics of OSS*, December 2000, online www.idei.asso.fr/Commun/Conferences/Internet/Janvier2001/Papiers/Johnson.pdf

Kaven, Oliver: *Performance Tests: File Server Throughput and Response Times*,. PC Magazine, November 13, 2001, online www.pcmag.com/print_article/0,3048,a%253D16554,00.asp

KBSt: *KBSt-Brief Nr. 2/2000: OSS in der Bundesverwaltung*,. KBSt, 2000, online linux.kbst.bund.de/02-2000/

McKelvey, Maureen (2001): *Internet Entrepreneurship: Linux and the dynamics of OSS*, CRIC, online les1.man.ac.uk/cric/dp44.htm

Khalak, Asif (2001): *Economic model for impact of OSS*, MIT, online opensource.mit.edu/papers/osseconomics.pdf

Klever, Ralf (1999): *die tageszeitung, Redaktionssystem*, Wizards of OS 1, online: mikro.org/Events/OS/ref-texte/klever.html

Köhntopp, Kristian et al (2000): *Sicherheit durch Open Source? Chancen und Grenzen" in „Datenschutz und Datensicherheit (DuD) 24/9 (2000)*,. Vieweg, Wiesbaden, S. 508–513, online www.koehntopp.de/marit/pub/opensource/KoeKP_00Sicherheit OpenSource.pdf

Köppen, A. / Nüttgens, M. (2000): *Open Source: Strategien für die Beratung*, In: Scheer, A.-W.; Consulting – Wissen für die Strategie-, Prozess- und IT-Beratung, Saarbrücken, pp. 231–242, online www.iwi.uni-sb.de/nuettgens/Veroef/Artikel/Consulting/OpenSourceStrategien.pdf

Krempl, Stefan (2001): *Studie empfiehlt Schily Server-Umrüstung auf Linux*, heise online News, 7. Dezember 2001, online www.heise.de/newsticker/data/jk-07.12.01-008/

Krempl, Stefan (2002): *Bundesrechnungshof fordert Einsatz von Open Source*, heise online News, 25. Februar 2002, online www.heise.de/newsticker/data/anw-25.02.02-004/

Krueger, Patricia (1999): *Tour de Source – A Guide to the Start Ups*, Wired, online www.wired.com/wired/archive/7.05/tour.html

Krüger, Wilfried / Bach, Norbert (2001): *Geschäftsmodelle und Wettbewerb im e-Business*, in Supply Chain Solutions – Best Practices im E-Business, hrsg. von Buchholz W./ Werner, H., Schaeffer-Poeschel, S. 29–51, online www.eic-partner.de/material/ebus.pdf

Lancashire, David (2001): *Code, Culture and Cash: The Fading Altruism of Open Source Development*, Firstmonday, online www.firstmonday.dk/issues/issue6_12/lancashire/index.html

Lerner, Josh / Tirole, Jean (2000): *The simple Economics of Open Source*, December 2000,

online www.univ-tlse1.fr/idei/Commun/Articles/Tirole/Simple-Economics.pdf

Lewis, Michael / Walch, Kameron / Whitehouse, Janelle (2001): *Managing Technological Innovation in the Wired Society: The Case of Linux Operating System*, Reality Bites, LLP,
online eplu.netgistics.com/PDF/1999-2000/Fall/BUSA541/Bytes.pdf

Lutterbeck, Bernd (2001): *Gutachten für das Expertengespräch „Softwarepatente und Open Source"*, Informatik und Gesellschaft, TU Berlin,
online www.ig.cs.tu-berlin.de/bl/061a/Lutterbeck-BT-6-2001.pdf

Mantarov, Bojidar (1999): *OSS as a new business model*, Dissertation, Reading,
online bmantarov.free.fr/academic/msc_essays/dissert.htm

Materna Unternehmensberatung (2001): *Ihre erfolgreiche E-Business-Strategie*,
online www.materna.de/Dateien/eBusiness/07220e-Business_RZ1.pdf

Miller, Robin (2001): *Learning from Mozilla's mistakes*, NewsForge, October 23, 2001
online www.newsforge.com/article.pl?sid=01/10/20/1841215&mode=thread

Monaco, Carol / Volpi, Ana (2000): *Linux Support Services Forecast and Analysis, 1999-2004*, IDC Doc #23598, Dezember 2000,
online www.idc.com/getdoc.jhtml?containerId=23598

Mercer Management Consulting (2001): *Managing the professional service firm*, Lehrstuhl Prof. Wolfgang Becker, Universität Bamberg,
online www.uni-bamberg.de/sowi/ufc/inhalt/download/studienunterlagen/fs_mmc.pdf

Mundie, Craig (2001): *The Commercial Software Model*, The New York University Stern School of Business, 3. Mai 2001,
online www.microsoft.com/presspass/exec/craig/05-03sharedsource.asp

Mustonen, Mikko (2001): *Copyleft – the Economics of Linux and other OSS*, Univ. of Helsinki,
online data.vatt.fi/antitrust/papers/m_mustonen.pdf

Nadeau, Tom (1999): *Learning from Linux*, OS/2 and the Halloween Memos. OS2Headquarters,
online pcsupport.about.com/gi/dynamic/offsite.htm?site=http%3A%2F%2Fwww.os2hq.com%2Farchives%2Flinmemo1.htm

N.N. (2001): *The Open Source Case for Business, Opensource.org*,
online www.opensource.org/advocacy/case_for_business.html

N.N. (2002): *SuSE will im zweiten Halbjahr profitabel sein*, Linux-Enterprise 07.02.2002,
online www.entwickler.com/news/2002/02/5505/news.shtml

Netcraft (2001): *Netcraft Web Server Survey, October 2001*,
online www.netcraft.com/survey

Nüttgens, M. / Tesei, E.: *Open Source – Konzept, Communities und Institutionen*, in: Scheer, A.-W. (Editor): Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 156, Saarbrücken 2000a,
online www.iwi.uni-sb.de/nuettgens/Veroef/Artikel/heft156/heft156.pdf

Nüttgens, M. / Tesei, E.: *Open Source – Marktmodelle und Netzwerke*, in: Scheer, A.-W. (Editor): Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft

158, Saarbrücken 2000b,
online www.iwi.uni-sb.de/nuettgens/Veroef/Artikel/heft158/heft158.pdf

Nüttgens, M. / Tesei, E.: *Open Source – Produktion, Organisation und Lizenzen*, in: Scheer, A.-W. (Editor): Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 157, Saarbrücken 2000c
online www.iwi.uni-sb.de/nuettgens/Veroef/Artikel/heft157/heft157.pdf

von Oberkulm, Peter Michael Fäs (2001): *Business Models des Electronic Commerce*, Lizentiatsarbeit,
online www.peterfaes.ch/lic/files/lpfkxay.pdf

Osterberg, Jürgen (2001): *Flotter Auftritt mit Zope*, Computerwoche Nr. 44/2001, p.26, October 19, 2001

O'Reilly / Autor N.N. (1999): *Open Source – kurz & gut*, O'Reilly & Associates

Porter, Michael (1995): *Competitive Strategy*, New York

Pomerantz, Gregory M. (2000): *Business Models for Open Source Hardware Design*, New York,
online pages.nyu.edu/~gmp216/papers/bmfosh-1.0.html

Prasad, Ganesh (2001): *Open Source-onomics: Examining some pseudo-economic arguments about Open Source*, Linuxtoday
online linuxtoday.com/news_story.php3?ltsn=2001-04-12-006-20-OP-BZ-CY

Quade, Jürgen (2001): *Qualität hat keinen Preis*, Computerwoche Spezial 2/2001, Computerwoche Verlag

Rasch, Chris (2001): *The Wall street Performer Protocol: Using Software completion Bonds to Fund OSS development*, First Monday, Vol. 6, No. 6, June 2001
online www.firstmonday.dk/issues/issue6_6/rasch/

Raymond, Eric (1998): *Homesteading the Noosphere*,
online www.tuxedo.org/~esr/writings/cathedral-bazaar/

Raymond, Eric (1998): *The Cathedral and the Bazaar*,
online www.tuxedo.org/~esr/writings/cathedral-bazaar/

Raymond, Eric (1999): *The Magic Cauldron*,
online www.tuxedo.org/~esr/writings/cathedral-bazaar/

Roehrl, Armin / Schmiedl, Stefan (2002): *Vogelfrei. Die wichtigsten Open Source-Lizenzen*, ct 1/2002, Heise Verlag

Robles , Gregorio (2001): *WIDI – Who Is Doing It? Knowing more about Developers*, Informatik und Gesellschaft Institute of the Technical University of Berlin,
online widi.berlios.de

Rosenberg, Donald K. (2000): *Open Source: The Unauthorized White Papers*, M&T Books, Foster City

Sandred, Jan (2001): *Managing Open source Projects: A Wiley Tech Brief*, John Wiley & Sons, Inc.

Schenk, Thomas: *Linux: Its history and current distributions*, IBM
online www.developer.ibm.com/library/articles/schenk1.html

Schmitz, Ludger (2001): *Der steinige Weg zu Linux-Standards*, Computerwoche, May 4, 2001

Shapiro, Carl / Varian, Hal R (1998): *Information Rules: A Strategic Guide to the Network Economy*, Harvard Business School Press,
online www.inforules.com

Shankland, Stephen (2001a): *VA Linux to sell proprietary software*, News.com,
online news.cnet.com/news/0-1003-200-6954900.html

Shankland, Stephen (2001b): *Sun acknowledges acquisition misfires*, News.com,
online news.cnet.com/news/0-1003-200-7383790.html

Spiller, Dorit (2001): *Free/Libre and Open Source Software: Survey and Study. OS/FS business models and best practices (working paper)*, Berlecon Research Berlin,
IDA/Unisys: Study into the use of OSS in the public sector. Part 1-3 Europäische Kommission, Unisys.,
online ag.idaprog.org/Indis35prod/doc/333

Security Space (2001): *Security Space Web Server Survey, October 2001*,
online www.securityspace.com/s_survey/data/200110/index.html

Sergio, G. (2001): *Got Linux? Many companies say no*, CNET, November 2001,
online news.cnet.com/news/0-1003-200-7803522.html?tag=mn_hd

Slywotzki, Adrian J./ Morrison, David J. (1997): *Die Gewinnzone. Wie Ihr Unternehmen dauerhaft Erträge erzielt*, Mercer Management Consulting, Verlag Moderne Industrie

Succi, Giancarlo / Paulson, James / Eberlein, Armin (2001): *Preliminary Results from an Empirical Study on the Growth of Open Source and Commercial Software Products*, Paper submitted for „Third International Workshop on Economics-Driven Software Engineering Research" (EDSER-3 2001),
online www.cs.virginia.edu/~sullivan/edser3/paulson.pdf

TechConsult (2001): *Fast jedes 5. Unternehmen setzt 2003 auf Linux*, Press Release TechConsult,
online www.techconsult.de/de/Services/studiendetail.cfm?id=82

Tuomi, Ilkka (2001): Internet, *Innovation and Open Source: Actors in the Network*, First Monday, Vol. 6, No. 1, January 2001,
online www.firstmonday.dk/issues/issue6_1/tuomi/index.html#author

Valloppillil, Vinod / Cohen, Josh (1998): *The Halloween Documents*, Open Source Initiative,
online www.opensource.org/halloween

Varian, H.R. (1997): *Versioning Information Goods*, Research paper prepared for Digital Information and Intellectual Property, Harvard University,
online www.sims.berkeley.edu/~hal/Papers/version.pdf

Varian, H.R. (1996): *Differential Pricing and Efficiency*, First Monday, volume 1, number 2,
online firstmonday.org/issues/issue2/different/

Varian, H.R. (1995): *Pricing Information Goods*, presented at the Research Libraries Group Symposium on „Scholarship in the New Information Environment". Harvard Law School, Cambridge, Mass,
online www.sims.berkeley.edu/~hal/Papers/price-info-goods.pdf

Välimäki, Mikko (2001): *Strategic Use of Intellectual Property Rights in Digital Economy – Case of Software Markets*, Helsinki Institute for Information Technology, online www.hiit.fi/de/hamilton.pdf

VDC (Venture Development Corporation, 2001) : *Linux's Future in the Embedded Systems Market (Summary for the Embedded Linux Community)*, online www.linuxdevices.com/articles

Vepstas, Linas (2001): *Is Free Software Inevitable?*, onlien linas.org/theory/freetrade.html

Wayner, Peter (2001): *Open source databases bloom*, Computerworld, September 9, 2001, online www.computerworld.com/storyba/0,4125,NAV47_STO63629,00.html

Weber, Steven (2000): *The political Economy of OSS,*. BRIE, online brie.berkeley.edu/~briewww/pubs/wp/wp140.pdf

Weiss, George (2001): *The Future of Linux and Open Source*, Gartner, Strategy & Tactics/Trends & Direction, Note Number AV-13-9850, June 20, 2001 online www.gartner.com/resources/98800/98894/98894.pdf

Weiss, George (2000): *OS Evaluation: Linux vs. Unix and Windows 2000*, Gartner Research Note, online www.gartnerweb.com/public/static/hotc/hc00091281.html

Wheeler, David (2001a): *More Than a Gigabuck: Estimating GNU/Linux's Size, Version 1.06, 30.6.2001, updated 8. 11. 2001*, online www.dwheeler.com/sloc/redhat71-v1/redhat71sloc.html

Wheeler, David (2001b): *Why OSS / Free Software (OSS/FS)? Look at the Numbers!*, online www.dwheeler.com/oss_fs_why.html

Whitlock, Nathalie Walker (2002): *The security implications of OSS*, IBM, March 2001,online www-106.ibm.com/developerworks/library/l-oss.html

Wiehr, Hartmut / Wild, Martin (2001): *Linux – Die Alternative*, Informationweek, July 7, 2001 and July 26, 2001, online www.informationweek.de/

Wieland, Thomas (2001): *Linux als Geschäftsfaktor*, Linux Enterprise, Linuxtag online www.cpp-entwicklung.de/downld/linuxtag2000_wieland.pdf

Wysong, Thom (2000): *Introduction to Open Source and Free Software*, online www.technodemocracy.org/papers/ossfs.html

Yager, Tom (2001): *BSD's strength lies in devilish details*, InfoWorld, November 2, 2001 online www.infoworld.com/articles/tc/xml/01/11/05/011105tcbsd.xml

Zerdick, Axel / Picot, Arnold / Schrape, Klaus / Artopé, Alexander / Goldhammer, Klaus / Lange, Ulrich T. / Vierkant, Eckart / López-Escobar, Esteban / Silverstone, Roger (1999): *Die Internet-Ökonomie. Strategien für die digitale Wirtschaft*, Springer

Kapitel 3

Technik

Einleitung

YACINE GASMI

Die große Wirtschaft hat die Open-Source-Bewegung lange Zeit für uninteressant gehalten, weil sie schlicht keine konkurrenzfähigen Produkte von einer Gemeinschaft erwartete, in der Software als Hobby programmierfreudiger Individuen und ohne unternehmerisches Dachwerk entstand. Open Source galt mehr als Spielwiese für Idealisten denn als Umfeld für professionelle Softwareentwicklung.

Durch den sichtbaren Erfolg zahlreicher Projekte, allen voran Linux, hat in den letzten Jahren jedoch ein Umdenken stattgefunden. Branchenriesen wie IBM oder SAP setzen zunehmend auf diesen alternativen Weg der Softwareentwicklung. Auf IBMs Inititiative hin entstand das Eclipse-Projekt, in dem namhafte Softwareunternehmen an der Herstellung einer Open-Source-Programmierumgebung arbeiten.[1] Und SAP, dessen Datenbank SAP DB seit Oktober 2000 unter einer Open-Source-Lizenz vertrieben wird, ist im Mai 2003 eine Partnerschaft mit dem schwedischen Unternehmen MySQL AB eingegangen, um gemeinsam eine Open-Source-Datenbank zu entwickeln.[2] Diesen Trend hin zu einer Öffnung gegenüber dem Open-Source-Gedanken verdeutlicht auch das Engagement diverser IT-Branchenführer in den *Open Source Development Labs (OSDL)*. Die OSDL wurden 2000 durch ein Konsortium der Unternehmen IBM, HP, CA, Intel und NEC gegründet. Zu den Mitgliedern zählen inzwischen über 30 namhafte Firmen aus dem IT-Sektor.[3] Durch die Unterstützung geeigneter Projekte mittels Bereitstellung von Entwicklungs- und Testumgebungen sollen Enterprise-Lösungen für Linux entwickelt und somit dessen Einsatz in Unternehmen forciert werden.

Die Beweggründe für das Engagement der genannten Firmen mögen vorrangig ökonomischer Art sein und sollen an dieser Stelle auch nicht diskutiert werden. Die genannten Beispiele zeigen aber, dass Freie Software[4] inzwischen ernst genommen wird, da man erkannt hat, dass diese Art und Weise der Softwareentwicklung durchaus qualitativ hochwertige Produkte hervorbringen kann. Wofür aber steht „Open

---

1 Die Homepage des Eclipse-Projekts findet sich unter http://www.eclipse.org.

2 Für die Pressemitteilung von SAP siehe Literaturverzeichnis: SAP 2003.

3 Die komplette Liste der beteiligten Unternehmen ist unter http://www.osdl.org/about_osdl/members/ zu finden.

4 „Freie Software“ und „Open-Source-Software“ werden hier stets im Sinne von quelloffener Software verwendet. Die genaue begriffliche Unterscheidung wird im Einführungskapitel ausführlich behandelt.

Source“ eigentlich genau? Der Begriff an sich benennt (auf Software bezogen) streng genommen lediglich eine Lizenzierungsform und ist in diesem Sinne keine Entwicklungsmethode. Die Offenlegung und die mit der spezifischen Lizenzierung verbundene freie Verfügbarkeit und Wiederverwendbarkeit des Quellcodes führen allerdings zu bestimmten Charakteristika bei der Entwicklung Freier Software. Beispielsweise ist das Mitwirken beliebig vieler Programmierer an einem Projekt nur dadurch möglich, dass jeder *im Prinzip* uneingeschränkten Zugang zum Programmquellcode hat. Betrachten wir deshalb an dieser Stelle der Einfachheit halber Open Source als ein bestimmtes Entwicklungsmodell und die proprietäre Variante der Softwareherstellung als das entsprechende Gegenmodell.[5] Dann liegt die Frage nahe, welches von beiden denn das bessere Modell ist. Oder wenn man etwas zugespitzt die Zukunft der Softwareentwicklung ins Spiel bringt: Wird sich ein Modell gegen das andere durchsetzen? Sofort ertönen die Prophezeiungen der ideologischen Hardliner auf beiden Seiten, die in der Open-Source-Bewegung entweder eine bald wieder verpuffende Modeerscheinung sehen oder entsprechend im anderen Lager das Ende der proprietären Softwareentwicklung verkünden. Die Wahrheit wird wie so oft irgendwo dazwischen zu finden sein.

Mit diesem Jahrbuch soll versucht werden, den Leser in seiner Urteilsbildung bezüglich der Möglichkeiten und Grenzen der Open-Source-Bewegung zu unterstützen. Die Worte „Möglichkeiten“ und „Grenzen“ sind ganz bewusst gewählt, denn letzten Endes spielen zu viele Faktoren eine Rolle im Entwicklungsprozess quelloffener Software, als dass ein Pauschalurteil über die Qualität von Open-Source-Software als solcher gefällt werden könnte.

Man tut sich ohnehin schwer, die Qualität von Software zu messen. Auf Grund ihrer Komplexität ist Software nicht greifbar und daher nur schwer zu beurteilen. Als Hilfestellung und Grundlage für eine Bewertung hat man bestimmte Anforderungen formuliert: Die wesentlichen sind Zuverlässigkeit, Wartbarkeit, Effizienz und Benutzerfreundlichkeit (Sommerville 2001, S. 28). Je nach Erfordernissen an die Anwendung fällt die Gewichtung unterschiedlich aus. Ein Banksystem beispielsweise sollte vor allem zuverlässig und insbesondere sicher sein, während bei einem Computerspiel die Benutzerfreundlichkeit einen hohen Stellenwert besitzt. Die genannten Qualitätskriterien bilden zwar eine Art Gerüst für die qualitative Analyse von Software, die letztendliche Bewertung eines Programmes wird dadurch aber nicht wesentlich einfacher. Einen Weg aus dem Dilemma bietet die Analyse und Bewertung des Entstehungsprozesses, d.h., man untersucht, wie die Software hergestellt wurde, und zieht dann Schlüsse auf die Qualität des Endproduktes. Dieser Ansatz hat zur Spezifizierung verschiedener Qualitätsstandards geführt – die bekann-

[5] Dies können wir ohne Gewissensbisse tun, da selbst Eric S. Raymond in seinem berühmten Artikel: „The Cathedral and the Bazaar“ so verfährt. Auch auf die unterschiedlichen herkömmlichen Vorgehensmodelle (z. B. Wasserfall-Modell) bei der Entwicklung proprietärer Software soll hier nicht näher eingegangen werden, vergleiche dazu (Sommerville 2001).

testen dürften die „ISO 9000"-Reihe[6] und das *Capability Maturity Model (CMM)*[7] sein – welche konkrete Anforderungen an den Entwicklungsprozess stellen.

In diesem Kapitel wollen wir analog verfahren und die Betrachtung der Freien Software unter technischen Gesichtspunkten mit einem Einblick in den Entstehungsprozess beginnen.

Den Anfang macht *Matthias Ettrich*. Als Gründer des *K Desktop Environment (KDE)* hat er den Aufstieg eines inzwischen internationalen Projektes mit über tausend freiwilligen Mitarbeitern hautnah von Anfang an miterlebt. Im ersten Beitrag dieses Kapitels schildert er den Entstehungsprozess quelloffener Software unter dem Aspekt der „Koordination und Kommunikation in Open-Source-Projekten." Die anfangs erwähnte Skepsis gegenüber quelloffener Software ist u.a. auf der Vorstellung begründet, dass durch das (scheinbar chaotische) Zusammenwirken sehr vieler Mitarbeiter, die zudem meist Hobbyprogrammierer sind, kein „professionelles" Produkt entstehen kann. Der Gedanke, dass sich die Vergrößerung des Teams in der Regel nicht förderlich auf den Entwicklungsprozess auswirkt, wurde bekannt als Brook's Law: „Adding manpower to a late software project makes it later." (Brooks, Jr. 1975; zit. n. González-Barahona 2003, S. 52). Open-Source-Projekte wie Linux schöpfen ihre Dynamik aber gerade aus der großen Anzahl an Beteiligten und ihren unteschiedlichen Kompetenzen. Um dieses Potenzial produktiv umsetzen zu können, müssen die vielen separaten Anstrengungen einzelner Personen koordiniert werden. Der Rolle der Projektleitung kommt dabei eine für den Erfolg des Projektes entscheidende Bedeutung zu, und so setzt sich Matthias Ettrich in seinem Artikel intensiv mit den Anforderungen an diese auseinander.

Als vielleicht wichtigste, da für das Projekt lebensnotwendige Aufgabe kann man die Motivierung der Entwicklergemeinschaft betrachten. Vergleicht man zwei der erfolgreichsten Open-Source-Projekte miteinander, Linux und Apache, dann stellt man fest, dass durchaus unterschiedliche Führungsstile zum Ziel führen können. Während Linus Torvalds, Gründer und Leiter des Linux-Projektes, den Dingen eher freien Lauf lässt, wird das Apache-Projekt von einem unternehmensartig organisierten Führungsgremium dominiert. Hier müssen sich Entwickler über lange Zeit bewähren, um in den elitären Kreis der Hauptentwickler aufgenommen zu werden, welcher für den größten Teil des geschriebenen Codes verantwortlich zeichnet. Der Großteil der Community ist grundsätzlich selten direkt an der Erarbeitung neuen Codes beteiligt. Ihr bedeutsamer Beitrag liegt vielmehr im Testen der Software sowie in der Fehlerfindung und -behebung. Ettrich nennt in seinem Text unterschiedliche Möglichkeiten der Mitarbeit und zeigt auf, welche Bedeutung das Internet für die Kommunikation zwischen den Projektteilnehmern erlangt hat. Abschließend kommt er noch auf ein heikles Thema der Open-Source-Bewegung zu sprechen, das Spalten eines Projektes:

---

6 Internationaler Industriestandard für die Entwicklung eines Qualitätsmanagementsystems in einer Organisation. Mehr zu ISO 9000 unter http://www.iso.ch/iso/en/iso9000-14000/index.html.

7 Von Carnegie Mellon's *Software Engineering Institute (SEI)* zur Verbesserung der Prozessqualität entwickelt. Bewertung des „Reifegrads" (engl.: *maturity*) einer Organisation nach 5-stufigem Modell. Näheres unter: http://www.sei.cmu.edu/cmm/cmm.html.

Da die Mitarbeit an einem Projekt freiwillig geschieht, kann ein unzufriedener Entwickler jederzeit abspringen. Die freie Verfügbarkeit des Quellcodes ermöglicht es ihm zudem, auf eigene Faust weiterzumachen und ein eigenes Projekt von dem ursprünglichen abzuzweigen. Dieses Vorgehen wird *forking* genannt und ist nicht ganz unproblematisch für die Open-Source-Gemeinschaft. Durch *forking* entstandene verwandte Projekte können sich im besten Fall zwar in völlig verschiedene Richtungen entwickeln, sie können aber auch zu Konkurrenten werden. Dem weit verbreiteten Argument, *forking* führe lediglich zu einer Art darwinistischem Selektionsprozess, bei dem sich die interessanten Projekte durchsetzen, stehen kritische Stimmen wie Nikolai Bezroukov (1999) gegenüber, die die Konkurrenzsituation für existenzbedrohend (für die gesamte Bewegung) halten – auf Grund begrenzter „Human Ressources", um die gebuhlt wird. Matthias Ettrich zeigt, welche Arten des *forking* existieren und welche Folgen aus ihnen resultieren.

Während Ettrich sehr praxisbezogen schreibt und dabei aus seinen Erfahrungen als Mitbegründer des KDE schöpft, zeichnet sich der zweite Beitrag dieses Kapitels durch eine stark wissenschaftliche Herangehensweise aus. Der Autor, *Gregorio Robles*, forscht und unterrichtet an der Universidad Rey Juan Carlos in Madrid und befasst sich seit einigen Jahren intensiv mit dem Thema „Libre Software"[8]. Sein Artikel schildert die Bemühungen, das Phänomen mit den Methoden und Werkzeugen der Softwaretechnik zu erfassen und somit greifbarer und verständlicher zu machen. Die große Frage lautet: Wie und wann wird ein Projekt erfolgreich? Denn derzeit lassen sich keine verlässlichen Voraussagen über den Werdegang eines Projektes treffen, es ist stets ein Spiel mit vielen Unbekannten. Diese Unsicherheit soll durch eine softwaretechnische Herangehensweise reduziert werden. Das Problem dabei ist, dass viele Konzepte der Softwaretechnik auf die spezifischen Arten, wie Freie Software entsteht, nicht anwendbar sind. Ausgehend von dem bekannten Basar-Modell von Eric S. Raymond als einem charakteristischen Entwicklungsmodell für Freie Software,[9] zeigt Robles die Möglichkeiten und Grenzen der softwaretechnischen Modelle und Verfahren auf diesem neuen Terrain. In seinem Beitrag „A Software Engineering Approach to Libre Software" veranschaulicht er, wo die Softwaretechnik im Bereich der freien Softwareentwicklung derzeit steht und welchen Weg sie in der Zukunft einschlagen muss. Er stellt die wesentlichen Analysemöglichkeiten vor und erläutert anhand einiger bereits durchgeführter empirischer Studien, welche Erkenntnisse bisher gewonnen werden konnten.

Ein wesentlicher Grund für den Einzug der Softwaretechnik, also das systematische Vorgehen bei der Softwareentwicklung nach bestimmten Modellen, war der Wunsch, Software fehlerfreier und somit zuverlässiger und sicherer zu machen.

Da wie schon erwähnt die bekannten Verfahren der Softwaretechnik auf die Analyse quelloffener Software nicht so ohne weiteres anwendbar sind, könnte man

[8] Libre Software wurde 1999 auf der IST-Konferenz in Helsinki als übergreifender Begriff für quelloffene Software vorgeschlagen, der sowohl Freie Software als auch Open-Source-Software abdecken sollte. Siehe dazu die Homepage der *European Working Group on Libre Software* unter http://eu.conecta.it/.

[9] Robles betont aber, dass die Begriffe Freie-Software-Entwicklung und Basar-Modell nicht gleichzusetzen sind.

leicht zu dem Schluss kommen, Freie Software sei im Vergleich zu proprietär entwickelter vielleicht unsicherer, qualitativ schlechter. Dennoch werben Anbieter Freier Software häufig mit dem Schlagwort Sicherheit, die sie geradezu als einen entscheidenden Vorteil gegenüber der Konkurrenz aus dem Closed-Source-Lager propagieren. Und es sind in der Tat nicht wenige, die den quelloffenen Weg der Softwareentwicklung als den Erfolg versprechenderen in puncto Softwaresicherheit ansehen. Ein wesentlicher Grund für den Sicherheitsgewinn läge gerade in der Offenlegung des Programmcodes, denn sie ermögliche es Interessierten, seien es nun Anwender oder Entwickler,[10] den erstellten Programmcode zu überprüfen. Dadurch könnten Fehler schneller entdeckt und behoben werden. Eric S. Raymond drückte diesen Sachverhalt in seinem Artikel *The Cathedral and the Bazaar* mit folgendem viel zitierten Satz aus: „Given enough eyeballs, all bugs are shallow" (1999, S. 41).

Doch diese Argumentation stößt nicht überall auf Zustimmung. Ein beliebtes Gegenargument ist, dass, gerade weil sehr viele Hände am Programmcode herumwerkeln, die Qualität des Produktes leidet. Für die einen kann ein Projekt also nicht genug Beteiligte haben, die anderen betrachten Software wiederum als sprichwörtlichen Brei, den zu viele Köche bekanntlich verderben, und so ist seit einigen Jahren eine rege Debatte zum Thema Open Source und Softwaresicherheit in den Medien zu verfolgen.

Robert A. Gehring, wissenschaftlicher Mitarbeiter im Fachgebiet Informatik und Gesellschaft der TU Berlin, analysiert im dritten Beitrag dieses Kapitels die laufende Diskussion. Seine Arbeit mit dem Titel „Sicherheit mit Open Source – die Debatte im Kontext, die Argumente auf dem Prüfstein" versucht zunächst aufzuzeigen, welche Faktoren die Diskussion dominieren und in welchen Bereichen die Debatte nur unzureichend geführt wird bzw. welche Argumente schlichtweg übersehen oder übergangen werden. Ausgehend von dieser Analyse fokussiert Gehring die Frage nach dem Potenzial von Freier Software bezüglich Softwaresicherheit auf die Bereiche Softwaretechnologie und Software-Ökonomie. Er bezieht in seine Argumentation aktuelle, bislang in der Debatte wenig oder nicht berücksichtigte Arbeiten ein und bietet dem Leser mit dem vorliegenden Werk ein Dokument, das einen neuen Zugang zur Thematik erschließen hilft.

Bislang wurde hier die Betrachtung auf die Entstehung und, unter dem Aspekt der Softwaresicherheit, auf bestimmte Eigenschaften von Freier Software fokussiert. Will man dem Thema Open Source aus technischer Sicht einigermaßen gerecht werden, kommt man aber nicht umhin, den Blick zu heben und auch über das Umfeld von Freier Software schweifen zu lassen. Welche Einflüsse wirken auf die Open-Source-Bewegung ein? Welche Auswirkungen hat diese auf andere „Geschöpfe" der Informationstechnologie? Ein sehr beeindruckendes Geschöpf der Informationstechnologie ist das Internet. Dass das Internet mit seinen Kommunikationsmöglichkeiten entscheidend zum Erfolg der Open-Source-Bewegung beigetragen hat, dürfte spätestens nach der Lektüre des Beitrags von Matthias Ettrich besser klar werden. Doch auch Freie Software hat wiederum ihren Anteil an der Erfolgs-

[10] Die Grenze zwischen Anwender und Entwickler verschwimmt bzw. verschwindet im Open-Source-Umfeld, da der Anwender eigenhändig Verbesserungen an der Software vornehmen kann. Näheres dazu im Beitrag von Matthias Ettrich.

story des WWW. Wer kann sich das Internet ohne offene Protokolle wie TCP/IP oder freie Skriptsprachen wie PHP oder Perl vorstellen? Mit der voranschreitenden Vernetzung ist auch das Bedürfnis nach Interoperabilität gestiegen. Software soll nicht länger nur als eigenständiges Programm auf einem lokalen Rechner laufen, sondern sich als Baustein in ein System aus verteilten Diensten integrieren. Doch wie erreicht man es, aus einem Dschungel von unternehmensspezifischen Implementierungen ein funktionierendes Ganzes zu bilden? Man schafft Standards. Das für das Internet zuständige Standardisierungsgremium ist das *World Wide Web Consortium (W3C)*. Seit 1994 hat das aus rund 400 Organisationen bestehende Konsortium so namhafte Bestandteile des Internets wie das *Hypertext Transfer Protocol (HTTP)* oder die *Extensible Markup Language (XML)* durch seine Standardisierungsverfahren geschickt und mittlerweile bereits über 50 Spezifikationen erstellt.

Mit dem Beitrag von *Dirk Kuhlmann*, Forscher an den *Hewlett Packard Laboratories* in Bristol, riskieren wir einen Blick über den semantischen Tellerrand des Begriffes „Open Source" und beschäftigen uns mit der Welt der Standards, genau genommen der *offenen Standards*. Kuhlmann geht in seinem Artikel „Open Source und offene Standards" der grundlegenden Frage nach, was Offenheit überhaupt bedeutet. Während bei Open Source die Offenheit, wie der Name schon sagt, in der Verfügbarkeit des Quellcodes begründet liegt, fällt es bei Standards weniger leicht, Offenheit konkret festzumachen. Ausgehend von dieser Fragestellung führt er den Leser auf unkonventionelle Art durch die Wirren der Standardisierungsmechanismen und schafft es immer wieder, Brücken zu verwandten Themen wie den Softwarepatenten oder der Kontroverse um die *Trusted Computing Platform Alliance (TCPA)* zu schlagen. Die Open-Source-Bewegung lässt Kuhlmann dabei stets im Hintergrund schweben, um sie an geeigneter Stelle ins Bewusstsein des Lesers zu rücken und deren Rolle oder Potenzial in den Fokus zu stellen.

Damit endet der kleine Exkurs in die Welt der offenen Standards, und zum Ende des Kapitels wird noch einmal ein praxisrelevanter Aspekt des Themas ins Blickfeld gerückt. Nicht zuletzt die Migration der Münchener Stadtverwaltung auf Open-Source-Software hat gezeigt, dass ein wachsendes Interesse bei Behörden und Unternehmen zu beobachten ist, auf Open-Source-Produkte umzusteigen. Als Beweggründe sind hierbei u.a. der Wunsch, nicht von einem Softwarehersteller abhängig zu sein, die geringeren Kosten einer freien Softwarelösung sowie mangelndes Vertrauen (ob zu Recht oder nicht, sei dahingestellt) in die Sicherheit proprietärer Software genannt worden (SPD-Fraktion im Münchner Rathaus 2003). Vielleicht trägt sich der eine oder andere Leser ebenfalls gerade mit dem Gedanken, in seiner Firma Linux als Betriebssystem einzuführen. Dann dürfte der letzte Beitrag dieses Kapitels mit dem Titel „Erfolgsfaktoren bei der Einführung von Linux in Unternehmen" von *Peter Ganten* besonders interessieren. Der Aufsatz klärt über Schwierigkeiten und Risiken auf, die bei einer Einführung von Open-Source-Software zu beachten sind. Da er sich im Kapitel „Technik" befindet, stehen dabei nicht finanzielle, sondern technische und strukturelle Aspekte im Vordergrund. Dennoch ist klar, dass bei einem solch gewichtigen Unterfangen wie der Umstellung der IT-Infrastruktur einer Firma oder einer Behörde die Frage der Wirtschaftlichkeit nicht außen vor gelassen werden kann.

Als Gründer und Geschäftsführer eines Unternehmens, das sich auf die Implementierung von (bzw. Migration zu) Linux spezialisiert hat, kann Peter Ganten aus den Erfahrungen zahlreicher Projekte für Firmen und Behörden schöpfen. In seinem Beitrag beschreibt er, wie die Planung und Durchführung einer Umstellung der Server und Clients auf Linux bei einem mittelständischen Unternehmen aussehen könnten. Dazu schildert er zunächst die Entwicklung einer Open-Source-Strategie und darauf aufbauend die letztendliche Umsetzung der Migration.

Die vorgestellten Beiträge spannen einen Bogen von der Entstehung bis zum konkreten Einsatz quelloffener Software und sollten dem Leser somit einen umfassenden Überblick über die technische Seite von Open Source vermitteln können. Da von verschiedenen Seiten gerne Pauschalurteile über die Qualität und das Potenzial von Freier Software gefällt werden, ist eine umfassende Kenntnis der technischen Hintergründe äußerst wichtig. In diesem Sinne: Eine erhellende Lektüre!

Literatur

Brooks Jr., Frederick P. (1975): *The Mythical Man-Month: Essays on Software Engineering*, Addison Wesley

Bezroukov, Nikolai (1999): *Open Source Software Development as a Special Type of Academic Research (Critique of Vulgar Raymondism)*, First Monday, volume 4, number 10 (October 1999), online http://www.firstmonday.dk/issues/issue4_10/bezroukov/ (25.1.2004)

González-Barahona, Jesús M. und Robles, Gregorio (2003): *Free Software Engineering A Field to Explore*, in: Upgrade Vol. IV, No. 4 (August 2003), S. 49–54, online http://www.upgrade-cepis.org/issues/2003/4/up4-4Gonzalez.pdf (25.1.2004)

heise online News, 30.8.2000: *Branchenschwergewichte investieren in Linux* online http://www.heise.de//newsticker/data/vza-30.08.00-000/ (25.1.2004)

Raymond, Eric S. (1999): *The Cathedral and the Bazaar*, S. 27–78, in: Eric S. Raymond: The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary, Sebastopol, CA: O'Reilly, online http://www.openresources.com/documents/cathedral-bazaar/main.html (25.1.2004)

SAP (2003): *SAP stellt ihr Datenbankprodukt der weltweit populärsten Open Source Community zur Verfügung*, Pressemitteilung vom 26.5.2003, online http://www.sap.com/germany/aboutSAP/press/press_show.asp?ID=1081 (25.1.2004)

Sommerville, Ian (2001): *Software Engineering*, 6. Aufl., Pearson Studium, München

SPD-Fraktion im Münchner Rathaus (2003): *Rathaus-SPD entscheidet sich für Linux*, Presseerklärung vom 26.5.2003, online http://www.spd-rathaus-muenchen.de/presse/press39-linux_030526.pdf

Koordination und Kommunikation in Open-Source-Projekten

MATTHIAS ETTRICH

Die Entwicklung von Software ist ein komplizierter und schwer vorhersehbarer Prozess. Komplexe Softwaresysteme neigen nicht nur dazu, fehlerbehaftet zu sein, sondern in der Herstellung auch teurer zu werden, als erwartet, und immer hinter ihrem Zeitplan zurückzuliegen. Die Antworten der Softwaretechnik heißen Prinzipien, Methoden, komplexe mehrphasige Systementwicklungsmodelle und organisierendes Projektmanagement. Umso erstaunlicher sind die technischen Erfolge in der freien Softwareentwicklung. Die offensichtlichen Vorteile sind der fehlende Zeitdruck und die fehlenden Kosten. Doch auch das ist kaum hinreichend, um zu erklären, wie ein loses, weltweites Netzwerk von Hackern ohne erkennbare Planung und kaum erkennbarer Organisation es schafft, Softwaresysteme im Millionen-Zeilen-Bereich zu erstellen, zu pflegen und weiterzuentwickeln – und das unbezahlt und in ihrer Freizeit. Wie schaffen es freie Projekte, ohne autorisierte Projektleitung und Management, notwendige Entscheidungen zu treffen und die anfallenden Aufgaben sinnvoll zu verteilen?

1. Projektleitung: Entscheidungsfindung und die Rolle der Maintainer

Das Grundprinzip der Entscheidungsfindung in freien Softwareprojekten ist einfach: Diejenigen, die die Arbeit tun, entscheiden. Im Umkehrschluss bedeutet das, dass jeder Teilnehmer selbst entscheidet, woran er oder sie arbeiten will, und das dann auch tut. Wenn jeder genau das macht, was er oder sie will, warum kommt dann etwas zustande, das von außen betrachtet klare Züge von Strategie und richtigen Entscheidungen trägt und normalen Endanwendern zugute kommt? Oder umgekehrt gefragt: Wenn alle nur machen, was sie wollen, und keine kontrollierende und leitende Instanz existiert, warum gibt es dann nicht endlose Diskussionen und ein heilloses Durcheinander? Letzte Antworten auf diese Fragen lassen sich ohne gründliche und systematische Forschung nicht geben. Derzeit wissen wir, dass es funktioniert, ahnen aber lediglich, warum. Hier sind einige lose Gedanken, die zum besseren Verständnis des Entscheidungsprozesses in freien Softwareprojekten beitragen können:

- In jedem Betrieb und in jeder Arbeitsgruppe gibt es Mitarbeiter, die so gut sind, dass man sie weder beaufsichtigen noch herumkommandieren muss. Sie wissen, was zu tun ist. Sie sind kompetent, sie sind motiviert und sie sind der Traum eines jeden Chefs. In jedem erfolgreichen Open-Source-Projekt gibt es solche Mitarbeiter: Überdurchschnittlich intelligente, kreative und nicht zuletzt gebildete Menschen mit einem breiten Interesse an Informationstech-

nologie und ihren Anwendungen. Zwischen ihnen entsteht ein Gleichgewicht, das dazu führt, dass jede oder jeder genau an den Dingen arbeitet, die er oder sie am besten kann. Projekte sind nur dann erfolgreich, wenn sie von solchen Mitarbeitern getragen werden, und sie ziehen solche Mitarbeiter an, wenn sie erfolgreich sind. Das ist vergleichbar mit anderen Teams. Eine Fußballmannschaft ist nur dann wirklich gut, wenn sie gute Spieler hat. Umgekehrt wechseln gute Spieler am ehesten zu einer guten Mannschaft. Gute Teams werden damit besser, mittelmäßige oder schlechte Teams schlechter. Es wäre falsch zu behaupten, alle Entwickler im Open-Source-Umfeld seien technisch überragend. Nur ein Bruchteil aller gestarteten Open-Source-Projekte ist wirklich erfolgreich und schafft es, hinreichend neue Entwickler und Anwender zu begeistern. Die sich durchsetzenden Projekte aber schaffen es unter anderem deshalb, weil sie von wirklich guten Entwicklern getragen werden.

- In einem technischen Softwareumfeld trifft man selten Entscheidungen; man findet Lösungen. In einem solchen Lösungsfindungsprozess bewertet man Alternativen und wählt die beste, oder zumindest die jeweils am besten realisierbare aus. „Richtig" und „falsch" werden innerhalb eines solchen logischen Bezugssystems nahezu objektive Begriffe. In einer Welt, in der sich im allgemeinen leicht beweisen lässt, wer recht hat, und alle Parteien offen für logisch korrekte Beweisführungen sind, gibt es naturgemäß wenig Konflikte.
- In Fragen in denen keine Einigung erzielt werden kann, weil unterschiedliche Werte, unbeweisbare Annahmen oder verschieden geartete Zielsetzungen dies verhindern, wird ein evolutionärer Ansatz verfolgt: Jede Gruppe geht ihren eigene Weg, bis entweder objektive Kriterien das erfolgreichere Konzept ausweisen oder sich ein Weg als Einbahnstraße herausstellt. Schließen sich zwei Konzepte gegenseitig aus, wird das erfolgreichere das unterlegene Konzept verdrängen, weil es mehr Anwender und letztendlich mehr Entwickler anzieht. Ungesunde Konflikte werden damit durch gesundes Konkurrieren gelöst.

Ähnliche Konzepte lassen sich in traditionellen Softwareunternehmen nur begrenzt umsetzen. Entscheidungen und Strategie sind hier nicht auf rein technische Kriterien beschränkt, sondern müssen eine Vielzahl weiterer Faktoren berücksichtigen. Technologie muss nicht nur nützlich und damit prinzipiell verkaufbar sein, sondern sie muss von dem jeweiligen Unternehmen in dessen spezifischem Marktsegment gewinnbringend absetzbar sein. Softwareentwicklung ist teuer, also kann ohne strategische Planung nicht einfach ins Blaue hinein programmiert werden. Wo es aber keine bezahlten Arbeitsplätze gibt, kann auch keiner auf Grund einer strategischen Fehlentscheidung seinen Arbeitsplatz verlieren. Und wo Softwareentwicklung nichts kostest, wird durch langwierige und später verworfenene Fehlentwicklungen kein Geld verschwendet. Planung und Strategie können damit risikofrei durch Versuchen und Ausprobieren ersetzt werden, Konstruktion wird zu Evolution.

Hierzu kommen die hohe technische Qualifikation, die Lernbereitschaft und die Motivation der Mitarbeiter, die zu einer Produktivität führen, die über dem in-

dustriellen Durchschnitt zu liegen scheint. Zwei Dinge mögen hier entscheidend sein: Erstens leisten Menschen mehr, wenn sie zu 100% hinter den ihnen aufgetragenen Aufgaben stehen und selbst Entscheidungen treffen dürfen. Zweitens findet ein natürlicher Auswahl- und Anpassungsprozess statt. Ein unqualifizierter Kollege in einem Open-Source-Projekt kann nach einem missglückten Anlernversuch ohne Konsequenzen einfach freundlich ignoriert werden, ausgebrannte Kollegen verlassen das Projekt schon alleine dadurch, dass sie nichts mehr beitragen. Wenn ein Teilprojekt keinen Spaß mehr macht, wird es liegen gelassen, bis es ein anderer übernimmt, dem es wirklich Spaß macht. Solche Möglichkeiten haben Unternehmen nicht, sie müssen sich der sozialen Verantwortung stellen, die mit der vertraglichen Bindung zwischen Arbeitnehmer und Arbeitgeber einhergeht, und sie haben eine Verantwortung ihren bezahlenden Kunden gegenüber.

Ganz ohne Struktur kommt jedoch auch ein freies Projekt nicht aus. Diese Struktur besteht aber nicht aus vordefinierten Rollen mit Arbeitsbeschreibungen, die in einem geordneten Verfahren mit den am besten geeigneten Kandidaten besetzt werden. Sie entsteht vielmehr dynamisch aus der Gruppe heraus, der natürlichen, sozialen Veranlagung des Menschen folgend. Genauso wie eine kleine Gruppe Menschen in der Lage ist, gemeinsam zu spielen, zu jagen, zu kochen oder einen Frühjahrsputz in der Wohnanlage durchzuführen, so ist es auch möglich, gemeinsam Software zu entwickeln. Jeder sucht sich die Rolle, die am besten zu ihm oder ihr passt, und einige übernehmen automatisch eine Führungsrolle in einem gewissen Bereich, getrieben vom Charakter und gestützt auf durch Arbeit und Wissen erworbenen Respekt der Mitmenschen. Eine solche auf persönliche Relationen gestütze Selbstorganisation skaliert natürlich nicht. Dass es bei größeren Softwareprojekten dennoch funktioniert, verdanken wir einer besonderen Eigenschaft guter Software: Sie ist modular aufgebaut, aus relativ unabhängigen Teilen. Große freie Softwareprojekte folgen der Struktur der Software. Für jedes unabhängige Teil gibt es in der Regel auch ein unabhängiges Teilprojekt, das sich nach den oben genannten Prinzipien selbst organisiert.

Eine Rolle, die sich in allen Teilprojekten herauskristallisiert, ist die Rolle des Verwalters oder *Maintainers*. Diese Entwickler führen alle Fäden zusammen und tragen letztlich die Verantwortung. Je nach Persönlichkeit des Betreffenden kann sich die Tätigkeit im Einzelfall zwischen aktiver Projektleitung oder reaktiver Serviceleistung bewegen. Die ersten Maintainer sind immer die ursprünglichen Autoren des Codes. Später, wenn sich diese anderen Aufgaben zuwenden, werden es in der Regel diejenigen Entwickler, die am aktivsten am Code arbeiten. Zu den Aufgaben der Maintainer gehört das Überprüfen von Code-Beiträgen, das Sicherstellen der Funktionsfähigkeit, das Ausarbeiten und Einhalten eines Releaseplanes und das Kommunizieren mit Mitentwicklern. Wenn Entwickler Codeänderungen vorschlagen bzw. Änderungen in Form von Patches einreichen, hat ein Maintainer die Möglichkeit, diese anzunehmen, für eine spätere Version zurückzustellen, oder sie zurückzuweisen. Dies ist eine sehr wichtige Aufgabe und mit einer großen Verantwortung verbunden. Der falsche Mann oder die falsche Frau auf einer solchen Position kann einem Projekt viel Schaden zufügen. Umgekehrt kann der richtige Maintainer viel Gutes für ein Projekt tun. Einen Patch zurückzuweisen kann auf eine Art gesche-

hen, bei der sich der Entwickler selbst zurückgewiesen fühlt. Oder es geschieht auf eine Art, bei der der Entwickler mit Freuden den Patch überarbeitet. Lässt man die technischen Aufgaben beiseite, gleicht die Rolle des Maintainers der Rolle eines Personalmanagers. Es geht darum, die Mitarbeiter am Erfolg des Projektes teilhaben zu lassen, sie fühlen zu lassen, dass das Projekt allen gehört. Und es geht darum, neue Entwickler für das Projekt zu gewinnen. Das geschieht beispielsweise dadurch, dass „dumme" Fragen nicht belächelt, sondern beantwortet werden, und dass unerfahrene Entwickler nicht schief angeguckt und ausgegrenzt werden, sondern ihnen geholfen wird, Erfahrungen zu sammeln und besser zu werden. Zusätzlich nehmen manche Maintainer auch die Rolle eines Schlichters wahr, falls es einmal zu emotional geladenen Diskussionen im Team kommen sollte.

Bei funktionierenden Projekten sind die Maintainer die am besten für diese Aufgabe geeigneten Entwickler. Sie sind diejenigen, die in den Augen ihrer Mitentwickler am meisten über den Code wissen, am aktivsten daran arbeiten und auch in nicht-technischen Fragen rund um das Projekt den Respekt ihrer Mitentwickler genießen. Der Grund ist basisdemokratisch und einfach: Ein schlechter oder inaktiver Maintainer wird schnell durch einen anderen ersetzt, im schlimmsten Fall über das Spalten des Projektes, auch *forking* genannt. Die Mechanismen sind dieselben wie bei einer Hobbyfußballmannschaft auf einem Bolzplatz: Benimmt sich der Kapitän daneben, spielt er schlecht oder bezieht er andere Spieler nicht genug ein, so übernimmt entweder ein anderer den Job, die Spieler wechseln zu einer anderen Mannschaft oder sie machen ihre eigene Mannschaft auf.

Bei aller Analyse bleibt die einfache Einsicht, dass Softwareentwicklung erstaunlich reibungslos funktioniert, wenn Softwareentwickler das Sagen haben. Das ist an sich nicht verwunderlich: Mathematiker, Informatiker, Physiker und andere natur- und strukturwissenschaftlich ausgebildete Akademiker stellen eine mathematisch-logische Elite, der die Verständigung mit ihresgleichen relativ leicht fällt. Erstaunlicher ist, dass viele Unternehmen dieses wertvolle intellektuelle Kapital ungenutzt zu lassen scheinen und keinen technischen Karriereweg für produzierende Ingenieure anbieten. Die Comic-Serie „Dilbert", von allen mir bekannten Programmierern gerne gelesen, zeugt davon. Programmierer werden als bloße Implementierer eingesetzt, während strategische Entscheidungen auf sogenannten „höheren" Ebenen getroffen werden, dominiert von Leuten mit ein- oder zweijährigen MBA-Ausbildungen oder ehemaligen Programmierern, die den Kontakt zum Produkt und seiner Entwicklung längst verloren haben. Die Freie-Software-Gemeinschaft stellt den radikalsten Gegenentwurf zur dilbertesken Ingenieursausbeutung dar: Eine Gemeinschaft, basierend auf geteiltem Expertenwissen und gemeinsamen Interessen, in der Rang nur durch technisches Wissen und Taten erlangt und erhalten wird.

Mit fröhlichem „Draufloshacken" in vielen kleinen Teams ist es aber nicht getan. Software will auch qualitätsgesichert, dokumentiert, lokalisiert und veröffentlicht sein. Was in einem kleinen Team noch ein relativ überschaubarer Prozess ist, stellt sich in größeren Projekten als fast übermenschliche Herausforderung dar, zumal Softwarentwickler vor geplanten Releases gerne dazu tendieren, „noch eben schnell" ein neues Feature einzubauen, das „ganz bestimmt" nichts kaputtmacht.

2. Planung und Projektkoordination

Planung in Open-Source-Projekten entsteht nicht zuletzt aus der Notwendigkeit koordinierter Releases. Software durchläuft einen Reifezyklus, der von einer ersten Veröffentlichung über diverse *Alpha-* und *Beta-Releases* zu einer endgültigen Version führt. In diesen Testphasen wird versucht, die Software einer möglichst breiten Anwendergruppe schmackhaft zu machen, um über frühe Rückmeldungen Fehler schnell beheben zu können. In einem großen Open-Source-Projekt mit dutzenden Teilprojekten und hunderten von Mitentwicklern ist es eine Herausforderung, die Test- und Veröffentlichungszyklen aller Teilprojekte zu synchronisieren, ohne ein allzu großes Maß an die Kreativität und Freude hemmender Bürokratie einzuführen. Verschiedene Projekte haben hier verschiedene Lösungen gefunden. Allen Lösungen gemein ist die Definition sogenannter Meilensteine, ab denen die Software „eingefroren" wird. Eingefroren bedeutet, dass nur noch Änderungen erlaubt sind, die dokumentierte Fehler beheben, nicht aber Änderungen, die neue Funktionalität hinzufügen.

Klassische strategische Entwicklungsplanung im Sinne von: „Wer arbeitet wann und woran?" findet kaum statt, kann aber mangels Weisungsbefugnis auch nicht stattfinden. Dies stellt eine interessante Herausforderung an das Entwicklungsmodell: Wie verhindert man auseinanderdriftende, inkompatible Entwicklungen? Was passiert, wenn verschiedene Entwickler oder Gruppen von Entwicklern parallel an verschiedenen Teilen der Software arbeiten, und sich Wochen später herausstellt, dass die Neuentwicklungen überhaupt nicht zusammenpassen? Betriebe leisten sich hierfür Projektleiter, vorausschauende Planung und, falls es nötig sein sollte, ein Meeting aller involvierten Entwickler. Diese Mittel stehen im verteilten Entwicklungsmodell im Open-Source-Umfeld in der Regel nicht zur Verfügung. Ziel ist es deshalb, Parallelität und den dadurch geschaffenen Bedarf an Kommunikation und Planung möglichst zu vermeiden. Dies glückt im Regelfall dank zwei einfacher Mechanismen:

- Alle Mitentwickler arbeiten immer mit der jeweils aktuellen Version des gemeinsamen Codes.
- Code-Änderungen werden möglichst früh und häufig in den gemeinsamen Code-Bestand eingespielt.

„Jeweils aktuelle Version" bedeutet hierbei nicht das letzte öffentliche Release, sondern der aktuelle, häufig erst wenige Stunden oder Minuten alte Code. Ein Arbeitstag beginnt in der Regel mit einem Softwareupdate auf die letzte Version und endet mit dem Einspielen der geschaffenen Änderungen. Dies legt dem einzelnen Softwareentwickler eine interessante Verantwortung auf: Die Software muss für Mitentwickler auch nach dem Einspielen der Änderungen funktionsfähig und damit einsetzbar sein. Ich kann zum Beispiel nicht größere strategische Umbauten vornehmen, wenn dies andere Entwickler über mehrere Wochen stark behindern würde, weil deren Software auf Funktionen meiner Software angewiesen ist. Das Entwicklungsmodell gleicht damit in gewisser Weise dem Straßenbau: Auch hier muss im laufenden Betrieb ausgebessert und erweitert werden. Muss eine Straße einmal für kürzere Zeit vollständig gesperrt werden, muss zunächst eine Umleitung gefunden

und ausgeschildert werden. Was zunächst als Nachteil erscheint, entpuppt sich in der Praxis als qualitätssichernder Vorteil. Softwareentwicklungsprozesse wie das populäre „extreme Programmieren“ (Wells 1999) fordern genau das gleiche: Jegliche Codeänderungen sollten am besten gleich, spätestens aber nach einem Tag in den Hauptquellbaum eingespielt und damit den Mitentwicklern zur Verfügung gestellt werden. Diese ständigen Updates helfen, böse Überraschungen inkompatibler Parallelentwicklungen zu vermeiden, und sie sorgen dafür, dass die Software auch während des Entwicklungszyklus niemals gänzlich auseinanderfällt, sondern jederzeit verwendbar ist.

Auch wenn die Software jederzeit funktionsfähig ist, und das Projekt es schafft, Releases und Testzyklen zu koordinieren, – wie stellt man sicher, dass nicht an den Bedürfnissen der Anwender vorbei entwickelt wird? Im traditionellen Softwareentwicklungsprozess kommen an dieser Stelle strategische Planungen ins Spiel, die sich auf Kundenuntersuchungen, Marktstudien, Rückmeldungen der Verkaufsflotte und nicht zuletzt auf die Erfahrungen der Chefs im Marketing, Verkauf und Produktmanagement stützen. Kein Unternehmen kann es sich leisten, eine neue Version ihres Hauptproduktes herauszubringen, die die Kunden so nicht haben wollen. Verständlicherweise wird alles versucht, dieses schlimmste Szenario im Vorfeld auszuschließen.

Freie Software hat keine bezahlten Spezialisten für Marketing, Verkauf oder Produktmanagement. Trotzdem stürzt sich eine große Benutzergemeinde auf jede neue Version, häufig noch vor der endgültigen Freigabe. Ein Grund hierfür ist sicherlich der günstige Preis von 0 Euro. Viel wichtiger aber ist die grundlegende Anwenderorientierung Freier Software. Aus Prinzip ist Freie Software im höchsten Maße an den aktuellen und dringlichsten Bedürfnissen der Anwender ausgerichtet, weil es die Anwender selbst sind, die die Software weiterentwickeln. An erster Stelle bei der Implementierung einer bestimmten Funktionalität steht fast immer der Wunsch, diese Funktionalität selbst nutzen zu können. Kommerzielle Softwarehersteller argumentieren manchmal, Freie Software sei etwas für Freaks und Technikverliebte, nicht aber für Endanwender. Das ist unaufrichtig argumentiert, denn auch Informatiker und andere naturwissenschaftlich ausgebildete Akademiker sind Endanwender, und es sind diese Endanwender, die für die Weiterentwicklung der Software sorgen. Was die Anwenderfreundlichkeit kommerzieller Software angeht, muss darauf hingewiesen werden, dass in einem kommerziellen Umfeld der Anwender nur indirekt auf die Software einwirken kann, nämlich durch seine Kaufentscheidung. Neue Funktionen werden dann implementiert, wenn sie sich verkaufen lassen oder dem Hersteller andere direkte Vorteile bringen. Im Umkehrschluss bedeutet dies, dass für den Anwender wichtige Funktionen nicht implementiert werden, wenn sich diese für den Hersteller negativ auswirken könnten. Offene Standards, definierte Protokolle, Zusammenarbeit mit anderen Produkten und die generelle Anpassungsfähigkeit der Software sind hier als Konfliktfelder zu nennen. Freie Software steht bei solchen Konflikten immer auf Seiten des Anwenders – eine Software, die von Anwendern für Anwender entwickelt wird. Dahingestellt sei, dass Freie Software manchmal die Hürden für Einsteiger etwas höher legt.

Doch wie schafft es ein freies Softwareprojekt, Anwender dafür zu begeistern, die Software selbst zu verbessern? Wie kommen Informatiker dazu, umsonst zu programmieren, statt als Berater gutes Geld zu machen? Und was treibt Anwender ohne Programmiererfahrung dazu, lieber programmieren zu lernen, um fehlende Funktionalität selbst implementieren zu können, anstatt einfach ein kommerzielles Produkt käuflich zu erwerben?

3. Die Projektgemeinde

In einem Open-Source-Projekt gibt es nicht nur Programmierer, sondern auch Tester, Grafikdesigner, technische Schreiber für die Dokumentation und Übersetzer, die die Software und die Dokumentation an verschiedene Sprachen anpassen. Die Übergänge sind fließend. So beginnen zum Beispiel viele Programmierer ihre Freie-Software-Laufbahn als Übersetzer. Das Gros der Mitarbeiter kommt immer aus der Anwendergemeinde der Software selbst. Übersetzer beginnen typischerweise als Anwender, die sich daran stören, dass Software nicht oder nur unzureichend lokalisiert ist. Zufriedene Anwender werden schnell zu Testern, weil sie sich für die neuesten Versionen interessieren. Fortgeschrittenere Tester haben Gedanken und Vorschläge, wie die Software zu verbessern sei. Über diese Vorschläge kommen sie in Kontakt mit den Entwicklern und lernen diese besser kennen. Über kurz oder lang entsteht der Drang, selbst Änderungen an der Software vorzunehmen. Aus anfänglich kleinen Patches werden schnell größerere Patches, und ehe man sich versieht, ist man Teil einer internationalen Community.

Daneben spielt aber auch die persönliche Anwerbung eine wichtige Rolle. Viele Entwickler kommen zur Freien Software, weil sie jemanden kennen, der bereits in einem Projekt mitarbeitet. Das können entweder alte Bekanntschaften sein, aus einem Leben vor oder außerhalb der Freien Software, oder aber neue Bekanntschaften, die auf Messen oder Kongressen gemacht wurden.

Diese Erklärungsmodelle sind aber nicht gänzlich hinreichend. Sie erklären, warum Entwickler dazu beitragen, ein Projekt zu vollenden. Sie erklären aber nicht, was Entwickler dazu treibt, ein neues Projekt anzufangen. „Ein neues Projekt" heißt in diesem Fall nicht unbedingt losgelöst von allen anderen Projekten. Alle größeren Softwareprojekte sind modular aufgebaut. Neben einem zentralen Kern bestehen sie aus einer Vielzahl kleinerer, relativ unabhängiger Teilprojekte. Wie eigenständige Projekte auch, bedarf jedes Teilprojekt zumindest eines Initiators, und einer großen Zahl Vollender. Initatoren werden nicht angeworben, sondern von ihrem inneren, kreativen Drang getrieben. Ein kurzer Artikel über ein erfolgreiches Softwareprojekt in einem Computermagazin kann schon ein Auslöser sein, diesem Drang freien Lauf zu lassen.

Neue Mitarbeiter in einem Unternehmen durchlaufen üblicherweise ein Trainingsprogramm. Sie bekommen einen Mentor als Ansprechpartner, der sie mit ihren Kollegen und Kolleginnen bekannt macht, mit der Unternehmensstruktur und den für sie relevanten Prozessen. In einem freien Projekt gibt es weder Einführungskurse noch professionelle Betreuer und schon gar keine Meetings mit physischer Anwesenheit. Trotzdem integrieren freie Softwareprojekte ständig neue Mitarbeiter aus al-

len Teilen der vernetzten Welt, die nach kurzer Anlernungszeit effizient und koordiniert zusammenarbeiten. Wie funktioniert diese Kommunikation?

4. Kommunikation in Open-Source-Projekten

Die Kommunikation unter den Teilnehmern eines Open-Source-Projektes steht auf zwei wichtigen Säulen: Dem Internet als schnelle und billige Möglichkeit, Daten auszutauschen und der englischen Sprache als international akzeptiertes menschliches Sprachprotokoll. Englisch hat sich als der kleinste gemeinsame Nenner klar durchgesetzt, als Sprache der Wahl auf Konferenzen, im Web, im Programmcode, für Dokumentation und für die Kommunikation der Entwickler untereinander. Das hat nichts mit falsch verstandenem Internationalismus à la Telekom-Werbesprüchen zu tun, oder gar einer Missachtung der eigenen Sprache. Die Gründe sind vielmehr rein pragmatischer Natur: Kein freies Projekt kann es sich leisten, Entwickler anderer Sprachgruppen auszugrenzen. Dazu ist das Interesse, an Freier Software zu arbeiten, einfach zu gering. Selbst in verhältnismäßig großen Ländern wie Deutschland oder Frankreich lässt sich national kaum eine kritische Masse von Teilnehmern erreichen. Und selbst wenn ein nationales Softwareprojekt technisch erfolgreich wäre – gegen die Konkurrenz eines internationalen Projektes, das Kompetenz aus allen Teilen der vernetzten Welt rekrutierten kann, hätte es keine Chance. Verständlichkeit steht dabei über sprachlicher Richtigkeit oder gar Sprachästhetik: die Mehrheit der Programmierer haben Englisch als zweite oder dritte Fremdsprache erlernt, zum Teil erst durch das Programmieren im Erwachsenenalter. Anglisten bzw. Linguisten mit Hang zur Pedanterie mag das missfallen. Freunde aktiver Sprachentwicklung und neuentstandener Kreoldialekte können dabei aber viel Spaß haben.

Neben Englisch setzen Open-Source-Entwickler eine Vielzahl von technischen Hilfsmitteln zur Kommunikation und Zusammenarbeit über das Internet ein. Das gemeinsame Grundmerkmal dieser Werkzeuge ist, dass sie wiederum frei sind. Dies ermöglicht allen teilnehmenden Entwicklern, die gleichen Werkzeuge zu benutzen und falls notwendig anzupassen und zu verbessern. Es ist aber auch eine politische Dimension dabei: Das Einsetzen nicht-freier Werkzeuge im Entwicklungsprozess wird allgemein als Makel empfunden und nur in Ausnahmefällen toleriert, bei denen keine langfristigen Abhängigkeiten geschaffen werden.

Die am häufigsten verwendeten technischen Werkzeuge sind:

E-Mail:

Persönliche E-Mail von einem Entwickler zu einem anderen ist ein unschätzbarer Kommunikationskanal. E-Mail ist schnell und kann für verschiedenste Arten von Daten, auch größerer Mengen, benutzt werden. Der Erhalt einer E-Mail stört nicht, und die Empfänger einer Nachricht können selbst entscheiden, wann und ob sie diese beantworten wollen.

Noch viel wichtiger als persönliche Nachrichten sind allerdings sogenannte Mailinglisten. Mailinglisten werden üblicherweise für ein bestimmtes Thema eingerichtet, das sich auch im Namen der Liste wiederspiegelt. Wenn als Beispiel für das KDE-Projekt eine Konferenz im tschechischen Nove Hrady in Südböhmen geplant

wird, geschieht dies auf einer Mailingliste *novebrady@kde.org*. Alle an der Konferenz Interessierten können die Nachrichten, die auf diese Liste geschickt werden, mitlesen und auch selbst Nachrichten an die Liste schicken. Für alle anderen wird üblicherweise ein elektronisches Archiv zur Verfügung gestellt, aus dem man sich über vorangegangene Diskussionen informieren kann.

Mailinglisten als Medium sind nicht ganz unproblematisch. Ihr größter Vorteil – die einfache Benutzung – ist auch ihr größter Nachteil: Jeder kann sie benutzen. Größere Projekte, oder Projekte die sich mit aktuell sehr interessanten Themen beschäftigen, leiden daher oftmals darunter, dass ihre Kommunkationskanäle von Beiträgen aus der Peripherie des Projektes überschwemmt werden, von Anwendern, beiläufig Interessierten oder einfach nur Leuten, die sich langweilen und gerne im Netz ihre Meinungen verkünden. Diese offenen Diskussionen sind wichtig, um das Interesse an einem Projekt zu fördern, können aber für die Arbeit des Projektes äußerst hinderlich sein. Wenn Entwickler jeden Abend nur ein bis zwei Stunden Zeit für Freie Software haben, möchten sie verständlicherweise die Hälfte dieser Zeit nicht mit dem Lesen von E-Mails verbringen, nur um die wenigen relevanten Beiträge herauszufiltern.

Das Verhältnis von relevanten zu irrelevanten Beiträgen wird als „signal/noise ratio" bezeichnet. Überwiegen die irrelevanten oder weniger relevanten Beiträge auf einer Mailingliste (niedrige „signal/noise ratio"), werden überlichweise weitere Mailinglisten eingerichtet, die sich mit spezielleren Themen beschäftigen (weniger Beiträge, dafür aber höhere „signal/noise ratio"). Der Schreibzugriff auf solche Listen kann technisch auf eine bestimmte Gruppe eingeschränkt werden. Oftmals entscheiden die Mitglieder einer solchen eingeschränkten Liste darüber, wer neu aufgenommen wird und welche Kritierien neue Mitglieder erfüllen müssen.

Internet Relay Chat (IRC):

IRC ist ein offenes Chat Protokoll, manchmal auch als *Instant Messaging* bezeichnet. Es erlaubt einer Gruppe von Teilnehmer, sich direkt, Satz für Satz, über Tastatur zu „unterhalten". Auf IRC-Servern gibt es hunderte von verschiedenen Kanälen, die sich jeweils einem Thema widmen. IRC wird oft benutzt, um sich besser kennenzulernen oder einfach nur die Zeit zu vertreiben. Es ist aber auch eine Möglichkeit, „mal eben schnell" eine Antwort auf eine Frage zu bekommen, oder technische Probleme direkt zu diskutieren, ohne stundenlang komplexe E-Mails austauschen zu müssen. Bei Gruppen mit mehr als 4 Personen funktioniert Chat im Allgemeinen besser als eine Telefonkonferenz.

World Wide Web (WWW):

Am World Wide Web, umgangssprachlich nahezu zum Synonym für das Internet geworden, führt auch für die Freie-Software-Gemeinschaft kein Weg vorbei. Jedes Open-Source-Projekt unterhält eine Webseite, über die Informationen rund um das Projekt erhältlich sind, einschließlich der zum Herunterladen des Quellcodes. Häufig werden auch Web-basierte Diskussionsforen zu Themen rund um das Projekt angeboten. Diese öffentlichen Diskussionstafeln wurden in den letzten Jahren so erfolgreich, dass sie das klassische „News"-Protokoll aus Usenet-Tagen fast voll-

ständig verdrängt haben. Neuerdings werden auch dynamische Webseiten verwendet, *Wiki* genannt, die von Betrachtern der Seite direkt im Webbrowser verändert werden können (Leuf und Cunningham 2001).

File Transfer Protocol (FTP):

Vor einigen Jahren noch waren FTP-Server die wichtigsten Adressen, um Freie Software auszutauschen. Zwar hat das Web diese Funktion weitgehend übernommen, aber noch gehört FTP nicht ganz zum alten Eisen. Insbesondere für große Pakete wird es auch heute noch gerne eingesetzt. Das Prinzip ist einfach: Jedermann kann Dateien anonym in das öffentlich schreibbare (aber nicht lesbare) „Incoming-Verzeichnis" heraufladen. Ein Verwalter überprüft diese neuen Dateien regelmäßig und verschiebt sie nach erfolgreicher Prüfung in öffentlich lesbare Bereiche des Servers. Das manuelle Prüfverfahren verhindert, dass Softwarepiraten die freien Server zum Austausch geklauter, kommerzieller Software missbrauchen können.

Source Code Management (SCM):

Ein zentraler Bestandteil eines jeden Softwareprojektes ist ein zentrales Depot, in dem der gesamte Code eines Projektes – Programmcode, Grafiken, Übersetzungen und Dokumentation – verwaltet wird. Ein solches Depot wird als *Source Code Management System (SCM)* bezeichnet. Das zur Zeit am häufigsten eingesetzte System heißt *Concurrent Versions System (CVS)* (Vesperman 2003), der wahrscheinlichste Nachfolger nennt sich Subversion (CollabNet). Vereinfacht gesagt, ermöglicht ein solches System einer großen Gruppe von Entwicklern, zeitgleich an demselben Programm zu arbeiten und die Beiträge der Mitentwickler nachzuvollziehen. Jeder kann jederzeit für jede einzelne Zeile Code feststellen, wer die Zeile eingefügt oder verändert hat, zu welchem Zweck und wann. Im Bedarfsfall ist es auch einfach, einmal gemachte Änderungen wieder rückgängig zu machen oder das gesamte Projekt so zu rekonstruieren, wie es zu einem ganz bestimmten Zeitpunkt ausgesehen hat. Verwalter einzelner Module können auch automatisch informiert werden, falls jemand Änderungen eingespielt hat, oder den Zugang auf eine bestimmte Gruppe von Entwicklern beschränken.

Infrastrukturservices:

Vor einigen Jahren war es für freie Softwareprojekte noch eine Herausforderung, die zur Entwicklung notwendige Infrastruktur bereitzustellen. Meistens wurden Netzwerk-Bandbreite, Plattenplatz und Prozessorleistung für die Server von universitären Einrichtungen „abgezweigt", häufig ohne ausdrückliche Zustimmung der Leitung. Heute ist dies kein Problem mehr. Verschiedene Organisationen und Firmen bieten Rundumpakete an, die freie Projekte kostenlos nutzen können. Beispiele für solche Paketlösungen sind SourceForge,[1] das deutsche BerliOS Projekt,[2] oder der Savannah-Server der Free Software Foundation.[3] Die Zahl der in diesen vier Systemen beherbergten freien Softwareprojekte geht in die Zehntausende. Da-

1 Die Homepage des SourceForge-Projekts befindet sich unter http://sourceforge.net.
2 Näheres zu BerliOS unter http://www.berlios.de.
3 Zu finden unter http://savannah.gnu.org.

runter sind zugegebenermaßen viele gescheiterte oder verwaiste Projekte, aber auch viele gesunde Projekte im Wachstum.

Neben den technischen Werkzeugen darf ein wichtiges Kommunikationsmedium nicht vergessen werden: Der persönliche Kontakt von Mensch zu Mensch. Die meisten erfolgreichen Open-Source-Projekte beginnen früher oder später, sich real auf Konferenzen zu treffen, oder sie nutzen Community Events wie Messen oder allgemeine Veranstaltungen zu Freier Software als Rahmen, um sich zu treffen. Die Erfahrung lehrt, dass solche Treffen zu einem merkbaren Produktivitätsschub im Projekt führen, neben dem positiven Effekt der intensiv geleisteten Design- und Entwicklungsarbeit auf den Treffen selbst. Wir führen das einerseits auf gesteigerte Motivation zurück, andererseits darauf, dass Kommunikation über elektronische Medien deutlich besser funktioniert, wenn man den Gegenüber persönlich kennt. Der einzige Nachteil dieser Treffen, das Ausgrenzen von Entwicklern die aus diversen Gründen nicht teilnehmen können, wird dadurch mehr als ausgeglichen. Es ist trotzdem wichtig für freie Projekte, einer möglichst breiten Gruppe von Mitarbeitern die Teilnahme zu ermöglichen, u.a. durch die Wahl günstig gelegener und billiger Austragungsorte und durch direkte finanzielle Unterstützung finanzschwächerer Entwickler. Für Firmen oder andere Institutionen sind Kongresse und Entwicklertreffen die wahrscheinlich einfachste und effektivste Art, Freie Software finanziell zu unterstützen, ohne in die Projekte direkt einzugreifen.

Auch die beste Kommunikation kann nicht verhindern, dass Uneinigkeiten unter den Teilnehmern entstehen können. Solange das Ziel dasselbe ist, besteht Vermittlungspotential. Wenn aber die Zielsetzungen in einer Gruppe zu sehr auseinanderdriften, sind getrennte Wege oftmals die einzige Alternative zu endlosen und lähmenden Diskussionen. Viele Projekte sehen sich im Laufe ihrer Geschichte vor einer solchen „Gabellung“ – zu Englisch: „fork“.

5. Forking

Mit *forking* bezeichnet man das Abspalten eines neuen Projektes von einem bestehenden. Ein *fork* gilt häufig als ein Schreckensszenario innerhalb der Freie-Software-Gemeinschaft, spaltet er doch auch die Entwicklergruppe. Zusammenarbeit einer größeren Entwicklergruppe, so die Theorie, sei effektiver als zwei kleinere Entwicklergruppen, die untereinander konkurrieren. Wie man dazu auch stehen mag, die Möglichkeit zu einem *forking* ist eines der grundlegendsten Rechte, die ein Anwender Freier Software hat. Software, die diese Möglichkeit nicht bietet, kann nicht als Freie Software bezeichnet werden. Die ständige Gefahr eines potentiellen *forkings* zwingt Entwickler auf gleicher und respektvoller Ebene zusammenzuarbeiten. Einzelne Mitarbeiter einer kommerziellen Firma können damit drohen, die Firma zu verlassen und vielleicht ein paar Kunden mitzunehmen. Unzufriedene Mitarbeiter eines Open-Source-Projektes hingegen nehmen nicht nur ein paar Kunden mit, sondern gleich die ganze „Firma“ und einen guten Teil der Mitarbeiter.

Als Konzept wird *forking* also nicht nur toleriert, sondern als Voraussetzung für Freie Software geradezu eingefordert. In der Praxis durchgeführte *forkings* können

aber durchaus als „feindlich" angesehen werden und die durchführenden Entwickler sich unbeliebt machen. Inwieweit diese Einstellung sich auf die Softwareentwickler selbst erstreckt oder aber nur von Anwendern und Fans Freier Software lautstark vorgetragen wird, kann ohne gründliche Untersuchung nicht geklärt werden. Persönlich sehe ich das Recht zu „forken" als zu wichtig an, um die Inanspruchnahme dieses Rechtes scharf zu verurteilen, auch wenn das *forking* gegen den Willen der ursprünglichen Entwicklerschaft durchgeführt wird. c*ode-forkings* können auftreten, wenn große Teile eines Teams unterschiedlicher Meinung sind und keine Einigung mehr möglich ist. Das hat meistens mit technischen Einschätzungen zu tun, oft aber auch mit den Persönlichkeiten und persönlichen Vorlieben der involvierten Entwickler. Eine technische Alternative zu c*ode-forkings* ist die erhöhte Konfigurierbarkeit der Software, oftmals unterstützt durch eine Komponentenarchitektur. Bei einer aus einzelnen Komponenten aufgebauten Anwendung stellt im Grunde genommen erst der Anwender über verschiedenartige Optionen die eigentliche Anwendung nach persönlichen Vorlieben zusammen. Er oder sie entscheidet individuell, welche Komponenten hineingehören und welche nicht. Die Entscheidungsfindung, welcher Weg denn jetzt der „richtige" sei, wird damit den Anwendern überlassen. Über Rückmeldungen der Anwenderschaft kann das Projekt dann die Voreinstellungen so wählen, dass die meisten Benutzer mit der Software zufrieden sind, ohne dass Anwender und Entwickler mit anderen Vorlieben ausgeschlossen werden.

Forks treten aber auch dann auf, wenn ein Projekt allmählich einschläft, d.h. wenn die Maintainer nicht mehr schnell genug auf Vorschläge und Anfragen von Seiten der Benutzerschaft reagieren wollen oder können. Ein neues Team kann hier einspringen, möglicherweise gegen den Willen der ursprünglichen Entwickler, aber wahrscheinlich im Sinne der gegenwärtigen Anwender. Ein solcher *fork* bedeutet in der Praxis ein allmähliches Auseinanderdriften der Codebasis und damit meistens eine dauerhafte Trennung der Projekte. Es gibt aber auch Fälle, bei denen das „geforkte" und erfolgreichere Projekt später als neuer Hauptzweig in das ursprüngliche Projekt zurückgeführt wird. der *fork* wird damit zur Kur, die frischen Wind in alte Projekte bläst. Ein wichtiges Beispiel hierfür ist das Herzstück aller Freien Software, die *GNU Compiler Collection (GCC).* Als das Projekt vor einigen Jahren ins Stocken geriet, entstand eine zweite Gruppe, die mit dem *Experimentellen GNU Compiler System* (EGCS) einen Fork ins Leben rief (Henkel-Wallace 2002). EGCS lieferte Funktionalität, auf die die Anwendergemeinde lange gewartet hatte. Das neue System konnte seinem Urvater damit schnell an Popularität den Rang ablaufen, und zwang so dessen Maintainer zum handeln. Schließlich wurden Anpassungen an der Projektstruktur vorgenommen und beide Softwarebäume erneut unter dem ursprünglichen Namen GCC zusammengeführt.

Eine andere Art von *forks* sollte aber nicht vergessen werden, nämlich sogenannte konzeptionelle *forks*. Hierbei wird nicht etwa die Codebasis übernommen und verändert, sondern die Projektidee. Dies kann im Großen oder im Kleinen geschehen. In einem Textbearbeitungsprogramm kann ein Entwickler sich beispielsweise dahingehend entscheiden, eine neue Komponente für die Rechtschreibprüfung zu entwickeln, anstatt mit den anderen Entwicklern an der bereits existierenden zu arbeiten. Über einen gewissen Zeitraum wird das Projekt dann zwei verschiedene

Komponenten für die gleiche Aufgabe besitzen, bis eine sich als die bessere herausstellt und für die Kerndistribution ausgewählt wird. Diese Art von Konkurrenz ist ein wesentlicher Bestandteil der freien Softwareentwicklung, auch wenn Außenstehende manchmal den Kopf schütteln, warum es Dutzende von IRC-Clients, Texteditoren oder E-Mail-Programmen geben muss. Die Antwort ist einfach: Weil es Programmierer gibt, die die nötige Neugier besitzen oder ein technisches Interesse daran haben, diese zu schreiben. Kooperation und Zusammenarbeit kommt erst an zweiter Stelle. Sie wird dann gewählt, wenn sie anderen, persönlicheren Zielen förderlich erscheint. Ein Projekt kann sich zum Ziel setzen, das am meisten benutzte, anwenderfreundlichste E-Mail Programm zu schreiben. Ein solches Projekt wird höchstwahrscheinlich mit anden Projekten und vor allem Anwendern kooperieren. Ein anderes Projektes kann es sich zum Ziel machen, ein E-Mail Programm zur eigenen Verwendung gänzlich alleine zu schreiben. Kooperation ist hier eher hinderlich. Die beteiligten Entwickler dafür zu kritisieren, wäre wie Freizeitmusiker dafür zu kritisieren, dass sie ihre Zeit damit verschwenden, Musikstücke einzustudieren und sich selbst vorzuspielen, statt mit einem Verlag und einer Musikhandlung zu kooperieren und einfach CDs zu kaufen. Am Ende des Tages geht es um ein wunderschönes Hobby und den Drang, etwas mit den eigenen Händen zu schaffen, beim Programmieren wie beim Musizieren.

6. Zusammenfassung

Freie Softwareentwicklung ist ein dynamischer, evolutionärer Prozess. Das Open-Source-Konzept garantiert nicht automatisch, dass qualitativ überragende Software in kürzester Zeit entwickelt wird. Auf jedes erfolgreiche Open-Source-Projekt kommen hunderte von Projekten, die scheitern. Damit ein Projekt erfolgreich ist, müssen viele verschiedene Faktoren zusammenkommen, einschließlich des richtigen Timings und einer gewissen Portion Glück. Die wichtigste Überlebensfrage ist dabei immer, inwieweit ein Projekt es schafft, genügend Entwickler anzuziehen und zu halten. Anwender sind für ein Projekt nur an zweiter Stelle entscheidend: Als Motivationsfaktor und als Basis, neue Entwickler zu rekrutieren.

Die Motivation der einzelnen Entwickler ist sehr individuell. Sie reicht vom Eigeninteresse an der fertigen Software über technisch-wissenschaftliche Neugier, z.B. wie eine bestimmte Art von Software funktioniert, bis hin zu ganz persönlichen Zielsetzungen, z.B. ob man selbst in der Lage ist, eine bestimmte Software zu schreiben. Manche wollen auch einfach nur Spaß haben.

Koordination und strategische Planung innerhalb freier Softwareprojekte beschränken sich im wesentlichen darauf, einer möglichst großen Anzahl von Gleichgesinnten eine Arbeitsumgebung zur Verfügung zu stellen, in der sie fair und effektiv kooperieren können. Unterschiedliche Projekte finden unterschiedliche Lösungen, wobei größere Projekte auf Grund des komplexeren Release-Prozesses einen höheren Bedarf an Verwaltung und Regeln haben als kleiner Projekte.

Kommunikation innerhalb der Projekte geschieht über modernste Internet-Technologien. Ohne Internet kann es keine Entwicklung Freier Software geben, da

lokale Gemeinschaften nicht in der Lage sind, die notwendige kritische Masse aufzubringen.

Literatur

Adams, Scott: *The Official Dilbert Website*,
online http://www.dilbert.com (29.11.2003)
CollabNet Inc.: S*ubversion.tigris.org, a compelling replacement for CVS*,
online http://subversion.tigris.org (29.11.2003)
The GCC Home Page,
online http://gcc.gnu.org (29.11.2003)
Henkel-Wallace, David (2002): *A new compiler project to merge the existing GCC forks*,
online http://www.goof.com/pcg/egcs.html (29.11.2003)
Irchelp.org: Internet Relay Chat (IRC) help archive,
online http://www.irchelp.org (29.11.2003)
The K Desktop Environment Homepage,
online http://www.kde.org (29.11.2003)
KDE.news,
online http://dot.kde.org (29.11.2003)
KDE Mailing Lists,
online http://www.kde.org/mailinglists (29.11.2003)
KDE FTP Mirrors,
online http://www.kde.org/mirrors/ftp.php (29.11.2003)
Leuf, Bo und Ward Cunningham (2001): *The Wiki Way: quick collaboration on the Web*,
Boston: Addison-Wesley,
online Information unter http://www.wiki.org (29.11.2003)
Vesperman, Jennifer (2003): *Essentials CVS*, O'Reilly & Associates.
World Wide Web Consortium,
online http://www.w3c.org (29.11.2003)
Wells, Don (1999): *Extreme Programming, a gentle introduction*,
online http://www.extremeprogramming.org (29.11.2003)

A Software Engineering approach to Libre Software

GREGORIO ROBLES

The challenge of libre[1] software is not the one of a new competitor producing, under the same rules, software in a faster and cheaper way, and with higher quality. Libre software differs from "traditional" software in more fundamental aspects, beginning with philosophical reasons and motivations, followed by new economic and market guidelines and finishing with a different form of producing software. Software engineers cannot ignore this and for some years now the investigation of all these aspects has intensified. This article tries to introduce the reader into the most relevant technical aspects of libre software from the perspective of the development process (dubbed the Bazaar model; Raymond 1997). From a more quantitative point of view, several perspectives will be offered: the study of source code, of human resources and production costs. Finally, some of the proposals to continue advancing in the engineering research of libre software are presented.

1. Introduction

Libre software has won, without any doubt, a big importance lately. Although its philosophical principles (and first code) date back to the 1980s,[2] it has not been until some ten years ago that, with the proliferation of Internet connections, it has shown itself as a sound development and distribution method – to the point that it is increasingly becoming a threat for the traditional software industry, the latter being based on a closed development model and a distribution method relying on collecting licensing fees.

The "revolution" in the technical and technological methods in some libre software projects – those whose development form has been named the Bazaar model – has drawn the interest of software engineers. They became increasingly curious, seeing how a completely different approach has spurred the production of high quality, and low cost, software in the course of the last years. Even though the most interesting elements for these researchers have been the technical ones, in the author's opinion, the philosophical aspects related to libre software cannot be left aside: con-

---

1 In an attempt to avoid any confusion regarding the meaning of *free* in *free software*, throughout this article, the term *libre software* is used instead. It was chosen because of its meaning pointing towards *liberation*, and not of merely being costless. The term Open Source is refused for its ignorance about the philosophical foundations of what free software meant in the first place.

2 In fact, libre software already existed before the eighties, historically predating proprietary software. Cf., for example, Campbell-Kelly (2003, S. 54). 'Throughout the 1950s, programs were perceived as objects without intrinsic value, or at best with value that were no market mechanisms to realize. *Users got 'free' software from a computer manufacturer, or they found it through a user group*; more often, however, they wrote it in house.' (emphasis added) It was in 1984, however, that Richard Stallman began pursuing the GNU project as an alternative to the proprietary way of software development, distribution and deployment (Levy 2001, S. 427).

cepts like community and ethics are of great importance. Interpreting and understanding the observations, and answering apparently technical-only questions, requires to take into account this particular philosophy of software development, as we will see later.

There exists an ample confusion that we should clarify at the very beginning of this text: libre software and the Bazaar model are not synonymous. The freedom in software is a legal aspect that becomes reflected in the software license containing the terms and conditions under which the author has published, and allows to distribute, his work. On the other hand, we can locate the Bazaar model in the technical area. It is a way of developing software that promotes opening the development process as much as possible, thus offering the possibility to participate in development and distribution to anyone who desires to.

It is well known that many software projects with a free license follow a Bazaar development model, but this does not mean that all of them do so. Also, without doubt, it is the freedoms in libre software that facilitates carrying out a Bazaar development model. Perhaps, granting these freedoms is even the only way to make the Bazaar possible. In any case, unless it is indicated otherwise, the libre software which we are going to consider in this article will be the one that is produced following a Bazaar model. From a software engineering point of view, this is the one that is of the highest interest.

Clearly, since we are only going to consider Bazaar-style projects, we focus on a subgroup of all libre software projects. It is important to point out that, by doing this, we do not address the great majority of libre software projects which, according to Healy and Schussman (2003), show little activity and are organized in an individual way. In comparison, libre software projects following the Bazaar model are less numerous, but have (in general) a larger number of developers and users as well, what explains its broad diffusion and gives it a bigger impact on the software world.

The structure of this article is as follows: first the characteristics of the Bazaar model will be presented. Next a series of mostly empirical studies that have tried to clarify how the Bazaar works in practice will be discussed. This part has been divided in three sections: source code related technical analysis, analysis of the human resources that take part in the production of libre software and, finally, economic analyses from the engineering point of view. In order to finalize, we try to anticipate the future evolution of libre software research, indicate tools and methods that already are beginning to be used and argue which could be the following steps.

2. Characteristics of the Bazaar

The Bazaar development model received its name from the famous article by Eric S. Raymond (1997). In his article, he compared the traditional form of producing software (dubbed the Cathedral-style model) to a way being used with "success" in several case studies of libre software development.

In the traditional model (the Cathedral), tasks and roles are clearly defined and derived from project goals specified in advance. The development takes place in a

centralized way, dividing participants into three groups: (1) people who dedicate themselves to the design of the system (the architects); (2) a group of people dedicated to the management of the project; and (3) a final group fulfilling the pertinent tasks of implementation. On the other hand, in the Bazaar model, roles are not clear and can, and do, vary during the lifetime of the project. The important property of this Bazaar model is the constant interchange of knowledge, a flow of ideas of which all take advantage, what causes the software to evolve at a faster rhythm. This interchange emerges ideally in a spontaneous way and keeps running without supervision.

Unfortunately, although there exists a clear consensus on what the Bazaar model is and what it is not, this is not a totally defined concept and therefore it is subject to subjective interpretations. There are frequently debates that arise on whether one project follows the Bazaar model or not.

While showing some similarities, it seems clear that the Bazaar development model does not propose a development methodology as extreme programming or the Rational Unified Process[3] do. The Bazaar style is formulated as a series of prescriptions – technically we could call them patterns – that should be applied in order to make a project "successful". The patterns that are to be used in a project depend on several factors, such as project size (software size and/or number of developers) or status (whether a project is in its early stages or an already consolidated project with a large user-developer base, etc.). The application of a specific pattern is not very well defined either; its use should be guided by common sense, i.e. adjusted to the circumstances of the project in question.

Through the last years, together with the evolution of libre software, the concept of the Bazaar model itself has been evolving continuously. For example, while in Raymond (1997) it was stated that projects owe their beginnings to personal initiative (generally a developer who discovered a technical necessity), time has shown how big corporations (e.g., Netscape and SUN Microsystems) changed their strategies and released the source code of some of their software products in order to benefit from the Bazaar model. Today, a growing number of libre software projects are the heirs of closed, proprietary ancestors.[4] Although it is evident that we are dealing now with different project types, it seems that there is a certain unanimity nowadays to consider both approximations as projects following the Bazaar model.

The patterns of the Bazaar model are presented with thoroughness in this book in the article by Matthias Ettrich, so it should be sufficient to give only a brief description of the most important ones:

- *Treat your users as co-developers*: That is to open the development process to the maximum, so that developers interested in the project can be integrated seamlessly. More co-developers equals the possibility (for the project) of evolving more quickly, of creating functionality at a faster pace, etc.

[3] Being strict, we should consider RUP as a meta-methodology from which software development methodologies can be derived.

[4] To name just two of the most important: OpenOffice inherited its code base from the proprietary StarOffice, and the Mozilla web browser began its existence as the Netscape browser.

- *Release early*. A first version of the software has to be released although limited in functionality as soon as there is something presentable and functional. This way, the possibility of finding co-developers interested in participating increases.
- *Frequent integration* (also known as *release often*): Taking small steps allows the project to evolve in an incremental way, at the same time enabling the parallel debugging, and frequent feedback, by co-developers and users. In addition, frequent integration avoids an expensive and tedious, and error-prone phase of integration at the end of each development cycle.[5] The release policy differs from project to project, and as shown by Ehrenkrantz (2003) it can be very complex for big projects.
- *Maintain several versions*: The goal is to not overload the more conservative users with less-tried (i.e. more buggy) versions of software. Thus, two coexisting versions exist, one which is for development (the unstable one) supplying all the latest functionality (i.e. less proved and debugged) and a production version (the stable one) for those who put emphasis on stability, rather than on functionality, of software.
- *Modularize to the maximum*: High modularization allows parallel development and code reuse. Unfortunately, code production is not as much parallelizable as deployment and debugging is.
- *Dynamic decision making structure*: Although it is generally assumed that the final decision corresponds to the developers that actually code, usually an organizational structure exists – sometimes informal, sometimes more formal – by means of which the strategic decisions are made. The organizational form depends on the project, on its technical facets (project size, base of users, number of developers,...) as well as on other causes (historical reasons, balance between interests,...). For instance, Linux counts on "a benevolent dictatorship" model, Linus Torvalds being the dictator who delegates some supervision tasks to his "lieutenants", whereas other projects trust formulas based on more democratic meritocracies (Apache, GNOME...) or comittees that represent all "families" in the development community (GCC Steering Commitee, etc.).

Until now, this article has shown a point of view of process engineering, a widely studied field in software engineering. Even so, the Bazaar development model has a component that makes it unpredictable, sometimes we could even speak about some inherent "magic" in it. "Success" seems to be very capricious: it is possible that, although using the same ingredients that made another project "successful", a new one will ultimately fail. It turns out that the Bazaar is sometimes hard to reproduce; even harder it is to predict in advance, whether the attempt is likely to succeed or is doomed to fail. In any case, all this means that we know very little of the

[5] The article by MacCormack (2001) presents overwhelming evidence for the relationship between a frequent integration-frequent release-fast feedback development model and the quality of the end-product.

Bazaar model and there has to be done a serious research effort to gain knowledge on it.

Luckily, and owed to the circumstance that the Bazaar model pushes the openness of software development to its maximum (especially using means of telematics), a vast amount of information about the projects are available – ready to be retrieved and evaluated. In addition to having access to the source code itself, we are able to obtain all the previous states of a project, e.g., from archives of older releases or from a versioning system like CVS. Further documented is a number of interactions that have taken place during software development such as postings to newsgroups and mailing lists, information from bug-tracking tools, program documentation, etc. The main goal of investigating the named resources could be expressed as to remove the "magic" from the Bazaar to make it an engineering method.

3. Quantitative analysis of source code

The availability of source code presents one of the evident advantages at the time of undertaking the study of libre software. Besides the pure code itself, a series of tools exist for supporting the coordination of the software producing development team. For example, projects usually have a central source code repository where the project's code is stored (the current version as well as all the previous ones). Whenever a developer makes a modification to his local copy of the source code, he will have to synchronize it with the repository in order to grant the other project members access to the modified code. Also, the coordination of bug tracking – notification and resolution – is usually handled by means of a (central) web application where the errors are stored and can be retrieved.

Mockus et al. (2000) presented one of the first studies on libre software projects that contained a complete description of the development process and the underlying organizational structures, including qualitative and quantitative evidences. In order to quantify aspects of the development process such as developer participation, source code size, responsibility of code, productivity, density of defects, and error resolution time intervals, the software change history (stored within the changelog of the source code repository) is scrutinized as well as information in the bug-tracking system.

As a consequence of their study, the authors put forward several hypotheses. First, the existence of a small development group (the core team) that develops most of the new functionality of a project was demonstrated. Usually this group would not be bigger than 10 to 15 persons. For such a small team, informal means of coordination are used. Depending on the size of the problem (project), a small group may not be sufficient to fulfill the task, and more developers have to be integrated into the process. In such a case, another series of coordination mechanisms are to be introduced (individual or group code ownership, etc.), even possibly dividing the project in two different (smaller) ones. A second conclusion goes that in Bazaar type projects the number of participants contributing patches (i.e. corrections of bugs) is an order of magnitude larger than the core group; and the number of those notifying bugs, again, is an order of magnitude larger than the one of

the "patchers". Third, the authors state that the time of response to user feedback is significantly shorter in Bazaar-style projects than in projects following more traditional lines. Fourth, compared to proprietary projects that had passed a comparable testing scenario, the defect density is inferior in libre software projects.

The fact that the software produced with the Bazaar model is less inclined to contain errors is also discussed by Challet and Le Du (2003). They developed a mathematical model for simulating software development and debugging processes in closed source and open source projects as well. It appears that in order to reach a density of defects as low as the one in libre software projects, closed source projects would require a high number of above-average qualified developers. The proposed model assumes that in principle users update their software with each new release. Each new release removes defects and, at the same time, introduces some new ones. Usually, more defects are removed than introduced, thus the rate of defects decreases (more or less) quickly in time. Frequent releases allow to have a process with fast feedback in the Bazaar model which can be compensated in the closed source development model only by employing developers that produce code with fewer defects.

Godfrey and Tu (2001) explored the evolution of the Linux kernel following the classical theory of software evolution (Lehman et al. 1997). The main conclusion of their article is that Linux, although being a large project (with more than two million of lines of source code), grows in a superlinear way while developers are geographically dispersed. By contrast, until now, it had been postulated that the growth of large systems tends to slow-down itself over time. When scrutinizing the tree structure of the Linux kernel more thoroughly, the authors realized that more than half of the code corresponded to device drivers. Device drivers, however, are relatively independent parts of code, hence parallel development is feasible without encountering too many problems. On the other hand, the part of code corresponding to the "heart" of the kernel is relatively small. Thus, the superlinear growth of Linux is due to its strong modularization – in technical aspects as well as in organizational ones.

In libre software the exhaustive analysis of code does not have to limit itself to the study of independent applications, as it has been the case in the previous studies. Complete collections of libre software exist that have been integrated and packaged so that the whole set works properly as an integrated environment. Such packages come together with source code and can be analyzed too. This is the case for GNU/Linux distributions that have served to popularize libre software in general, and Linux based systems in particular.

In the contributions of Wheeler (2000 and 2001) we can find a quantitative analysis of the source code[6] and the programming languages used in several versions of the Red Hat distribution. González-Barahona et al. (2001) followed with a series of articles dedicated to the Debian operating system. Using a libre software tool called

[6] These studies use the following definition for the concept of physical source line of code: 'A physical source line of code (SLOC) is a line ending in a newline or end-of-file marker, and which contains at least one non-whitespace non-comment character.' (Park, 1992).

SLOCCount[7], the studies offer some kind of "radiography" of these two popular distributions.

The studies of Red Hat and Debian show how the respective different economical nature[8] for these two distributions affects the resulting software environment. It is worth to point out that the mean package size in Debian practically remained constant throughout the last five years. The natural tendency of packages to grow (in functionality, i.e. in code size) is neutralized by the inclusion of smaller packages, whereas in Red Hat the mean package size is growing continually. The reason for this difference is that in Debian the main requirement to have a package included in the official distribution is that some volunteer exists who packs it, whereas the composition of the Red Hat product depends on the effort that a company has to dedicate to ship the next version. Thus, Red Hat has a smaller number of packages, but, in an attempt to attract the widest user base, it includes almost all the important applications of the libre software panorama, each application in turn having almost all the available features included (resulting in comparably large applications), while Debian reflects a widespread – and diverse – software "ecology".

The possibility to study the evolution of the different Debian versions over time brings to light some interesting evidence (González-Barahona et al. 2003b). It is possible to perceive the importance of the C programming language (still predominant but loosing appeal), whereas non-compiled scripting languages (Python, PHP and Perl), and the platform-independent Java, show an explosive growth. The "classical" compiled languages (Pascal, Ada, Modula etc.) are in clear recession. Also, certain characteristics of the included programming languages can be observed: languages usually have a number of source code lines per file that is more or less constant over time,[9] and object oriented languages usually have files with fewer source lines of code – one reason why a higher modularity can be supposed.

In order to finish this section dedicated to the empirical study of source code, some proposals made in Robles and González-Barahona (2003); German and Mockus (2003) should be commented. Both articles present a complete, although limited, application of a methodology for the analysis of projects embarking all the previously mentioned ones. Future investigations should include many other parameters that have not been considered yet, as for example the use of complexity metrics.

4. Analysis of human resources

The traditional development model (or Cathedral) knows anytime the human resources which it has for the accomplishment of tasks, or so it is assumed. In the Bazaar model – due to its openness and the intention to integrate anybody who

7 SLOCCount (http://www.dwheeler.com/sloccount) is a suite of programs for counting physical source lines of code (SLOC) in possibly large software systems. It can count physical SLOC for a wide number of languages. It can take a large set of files and automatically categorize their types using a number of different heuristics, and also comes with analysis tools.

8 RedHat is a commercial distribution, whereas the second one, Debian, is developed and distributed completely by volunteers.

9 With the exception of some cases such as the shell scripting language(s).

wants to collaborate – this aspect cannot be planned for. From the management point of view this raises manifold problems beginning with the impossibility to know the number of tasks that can be fulfilled to a greater uncertainty regarding estimations of development time. It is necessary, therefore, to know more about libre software developers. In this section we will see some studies, ordered chronologically, that have concentrated themselves with these questions.

The open software development taking place in a distributed way lead to the question where developers are actually located. Soon it became clear that this could be one of the possible measures of the penetration of libre software. Early studies were being focused on socio-political questions in some cases and on technological rivalry between Europe and the United States in others. Dempsey et al. (1999) was the first research in this aspect by means of the study of a Linux file repository at the Metalab archive[10].

To be included in this file repository, every package had to add a denominated Linux Software Map (LSM) file where information on the software was specified. Some data was of forced fulfillment, for instance a brief description of the program. The specification of the main author and its e-mail address where also mandatory.

From the inspection of the top-level-domain of the authors' e-mail addresses some surprising conclusions could be derived, for example the unexpectedly huge number of top-level-domains relative to Germany (second one after popular .com).

Evidently, the method used was very prone to be inaccurate – the existence of "generic" domains like .com, .net, .org or .edu that were located at the first positions should be noted, or the assumption that a piece of software had only one author. It was for reasons like these that other studies were needed.

We can find another approach to these questions in Ghosh and Prakash (2000). The primary target of this study was centered around collecting data that could support a thesis that first appeared in Ghosh (1998). Ghosh supposed the existence of a "gift economy" as the resulting product in libre software is a public good. Contributions to the development were investigated in an empirical way. The inspection process was performed automatically by a tool named CODD. CODD tracks the source code according to patterns of authorship. These data were collected and in a later step processed in order to prepare full statistics. The study included the analysis of more than 3,000 projects, containing more than 1 Gigabyte, or about 25 million lines of code.[11]

More than 12,000 different authors were found and it could also be demonstrated that the participation follows the Pareto law, also known as the 80:20 rule: 80% of the code correspond to 20% of the developers, whereas the 80% of remaining developers had a contribution of 20% of the code. Many later studies have confirmed and extended to other forms of participation the validity of this result (messages to mailing lists, bug notifications, etc.).

[10] Metalab changed its name recently to ibiblio. The Linux archives can be visited at http://www.ibiblio.org/pub/Linux/.

[11] In this case it is not specified if lines of binary code or lines of source code (as in SLOCCount) were computed.

In 2001 a group of students at the Technische Universität Berlin, attempting to gain better knowledge of libre software developers, carried out a combined study (WIDI 2001) comprising an Internet-based survey as well as the analysis of e-mail addresses from a libre software repository (a complete GNU/Linux distribution) with CODD as described above. A further source of information was the Debian developer database, which stores data on the volunteers who maintain the Debian distribution. The online-survey was filled up by more than 5,000 people from all continents – thus far the largest survey of libre software developers at all.[12]

Regarding the number of developers and also considering the ratio of developer per million inhabitants, the results obtained show a slight advantage of Europe in comparison with the United States. The findings also confirm those results of the study based on LSM entrances, since the positions of the countries in the four classifications correspond with it, being always headed by the United States and Germany. Another result derived from this study is the distribution of time developers spend periodically on libre software: the average occupation is about 10 hours, while the median lies between 2 and 5 hours a week (with 25% of all developers). Less than 20% of the developers answered that they would spend more than 20 hours a week developing libre software. A later study sponsored by the European Commission (FLOSS 2002) validates these results and deepens insights in other aspects.

Another empirical study that dealt with libre software developers was conducted by Krishnamurthy (2002). Therein an analysis of 100 mature projects hosted at SourceForge is made. The results show a great majority of these projects being conducted by a small number of developers (usually one). This leads the author to conclude that the term "community" is not so well suited for the world of libre software. Instead he suggests using the description of a "voluntary association". Capiluppi et al. (2003) and Healy and Schussman (2003) arrive at the same conclusions with similar studies, the first with more than 400 random selected projects from Freshmeat (a very popular libre software announcement site) and the second by means of an exhaustive study of all projects hosted at SourceForge. In any case, these three studies do not differentiate between libre software in general and software developed according to the Bazaar model as is done in this article.

One of the most recent studies on libre software developers, González-Barahona et al. (2003), demonstrates how the development teams change over time. For most of the studied projects – especially those following the Bazaar model – different generations of core developers can be identified. If this finding is correct, the often made assumption that "success" in libre software is mainly due to the leadership of a single person (known as "code god") is refuted.

5. Empirical analysis and libre software production costs

The estimation of (human and technical) resources and of the time that it takes to successfully finish a project has always been one of the big challenges in all fields of engineering. In short, resources multiplied by time can be translated into production costs. Software engineering has always been weak in giving valid estimations

[12] A further analysis of the WIDI data is currently under way.

to help manage these parameters. Even though in libre software such aspects possibly do not have that importance as in the closed business world, from an engineering point of view it would be interesting to be able to calculate these parameters. The results, then, could be compared to their counterparts of "traditional" software projects.[13]

Koch and Schneider (2000) analyzed the interactions taking place in a libre software project. Their information source were mailing lists and the versioning system repository of the GNOME "project". The most interesting aspect of this study is the economic analysis. Koch wanted to verify the validity of classical cost prediction methods (function points, COCOMO model etc.) and discovered the problems that their application entails. Although he admits that the obtained results (taken with precautions) could be partially adjusted to reality, he concludes that libre software needs its own methods and models. The known ones simply do not fit well the nature of libre software projects. Evidently, the ability to obtain many of the data related to the development of libre software allows to be optimistic.

Another of the implicit conclusions from Koch and Schneider (2000) can be extracted from the use of the Norden-Rayleigh curve. This curve is based on the original suppositions of Norden dedicated to the study of scientific projects. For Norden, this type of projects could be modeled like a series of indefinite problems (but of finite number) that require of human effort ("manpower") for their resolution. The number of persons that has to be dedicated to solve problems is approximately proportional to the number of problems still to solve in any moment in time. It also models the fact that the members of the team are constantly learning in time. Following this assumptions, at the end of the project the number of people working on the project should be small, since the number of problems is also small. It seems evident that in libre software we cannot use this model, at least not in a general way, because one of its conditions is to know the number of developers on which to base our assumptions. We also need to know the time these programmers dedicate to solve problems. Both factors are usually unknown in libre software.

Previously in this article, studies on the evolution of two GNU/Linux distributions, Debian and Red Hat, have been presented. Using models for cost calculation typically applied in "classical" software engineering, and with the supply of lines of code from libre software projects, we are in the position to give some estimations. We are going to make use of the Basic COCOMO model which was proposed at the beginning of the 1980s (Boehm 1981) and that has been extended and improved in the 1990s as COCOMO II. COCOMO allows to make an estimation of effort and development time for (proprietary) software projects carried out within the traditional waterfall model. From the calculated effort (counted in man-months) and the duration of the project (in months) the total cost of the project can be estimated.

[13] This means that the quantitative analysis of economic parameters we are considering in this section is not one about cooking-pot-markets (Ghosh 1998), which has found quite some attention among economists. The latter is concerned with aspects like the production of public goods by libre software developers. The discussion here is structured more along the lines of interest of a company that wants to produce libre software and take advantage of the Bazaar.

The following table shows the results in brief. It can be noted that the Red Hat and Debian distributions have been growing through time, the versions of the latter being the larger ones. In comparison, there are estimations that esteem Microsoft Windows 95 and Microsoft Windows 2000 having 15 MSLOC and 29 MSLOC respectively.[14] The effort required to create so much code is enormous (thousands of person-years), whereas for the estimation of the time for its accomplishment it assumes that packages are independent from each other and thus can be developed in parallel. That way, it occurs that the given time is the one that it would take to create the biggest project included in the distribution; and it is for that reason that Red Hat versions, while being smaller in size, take more time to develop than some of the Debian versions (actually being larger). Finally, the estimation of development costs for each distribution (development starting from scratch) is given.

Name	*Release*	*MSLOC*	*Effort (person-years)*	*Time (years)*	*Cost (Million USD)*
Red Hat 5.2	1998-10	12	3,216	4.93	434
Red Hat 6.0	1999-04	15	3,951	5.08	533
Red Hat 6.2	2000-03	18	4,830	5.45	652
Debian 2.0	1998-07	25	6,360	4.93	860
Red Hat 7.1	2001-04	32	8,427	6.53	1,138
Debian 2.1	1999-03	37	9,425	4.99	1,275
Red Hat 8.0	2002-09	50	13,313	7.35	1,798
Debian 2.2	2000-08	59	14,950	6.04	2,020
Debian 3.0	2002-07	105	26,835	6.81	3,625

Effort, development and total cost estimation for different Red Hat and Debian versions

Without any doubt, the numbers shown in the above table must be taken with prudent caution. The COCOMO model has been proposed for large industrial projects following (more or less) the cathedral model. Therefore, using the same formula for projects in which the development model is Bazaar-style is only the first source of mistakes, the combination of which could make the obtained estimations totally invalid. As a first albeit rough estimation, these numbers should be sufficient.

[14] These numbers have been taken from Wheeler (2000). As it may seem obvious to the reader, the source for these numbers is the own company that produces the software, Microsoft. Due to the proprietary nature of its software it is impossible to have them reviewed, even to know if only the operating system has been taken into regard or also other components as the Microsoft Office suite.

6. Looking into the future of libre software engineering

As it has been possible to see throughout this article, software engineering is centered around a series of concepts. Unfortunately, some of them are not clearly defined. Thus, language becomes a barrier to share information and facts. This is the case for concepts as the following ones (although the list should not be limited to them):

- Project: It is still not very clear how to define a libre software project. We do not know if we have to consider GNOME (a desktop environment) or one of its components like Evolution (a personal information manager) as a project. Evolution is certainly part of GNOME, but with a degree of sufficient technological and organizational autonomy so that it would make sense to consider it being itself a project.
- Success: The concept of *success* is very diffuse in libre software. In traditional software engineering we could say that a project is successful when the stated goal is reached without exceeding the estimated constraints of budget and time. In libre software similar constraints do not exist and there are no functionalities planned for. All functionalities to be included are the developed ones at the moment of the release. On the other hand, it seems that the idea of "success" that is widely used for assessing libre software projects assumes a project that has a large developer and user community – something that may depend much on the goals and aims of the project.
- Core group: The core group of developers is commonly assumed to comprise the most involved persons in the development of a project. There is, however, no exact definition of how much involvement (in terms of time, contribution of lines of code or other means etc.) is necessary to form a part of this group.

In libre software engineering there is lack of a taxonomy for projects allowing to group them according to their characteristics. This article has described thus far how the simple classification of Bazaar and Cathedral is too vague for the degree of knowledge we want to acquire about libre software and its development. Setting out to develop a better classification one does not, however, have to start from the ideas that have been exposed in Raymond (1997).

Instead, this article proposes the creation of a complete suite of tools for the analysis of libre software projects, preferably in an automatic or semiautomatic way. This suite could be used to obtain the classification that was proposed in the previous paragraph and would allow to deepen investigations of projects from different perspectives. As this article has shown, several tools exist that could be integrated into this suite.

As far as the human resources are concerned, until now all studies have been dedicated almost exclusively to the people who develop source code. The fact that there exist other important type of activities has been widely ignored. Experience shows that when working groups grow in regard to the number of participants, division of labor and specialization gain more importance. For instance, tasks of internationalization or localization of a program are usually not carried out by code de-

velopers but by others. Also, the increase in complexity of software and the growing demand for user-friendly GUIs implies that graphical designers become integrated in many projects.

Another of the aspects that researchers are beginning to approach is the analysis of social networks formed by the developers. Several studies were directed in order to expose these social networks within the world of libre software. One main goal is to identify which developers or projects are in key positions within the community (Ghosh 2003; Madey et al. 2002).

Finally, it is important that models that can cope with the inherent complexity of the development process of libre software are created. Such models would allow to understand – and to predict – certain organizational aspects. That could be a first step on the way towards the use of intelligent agents initiating the creation of desired behaviors within a development process. There are already several projects aiming at doing exactly that.[15]

7. Conclusions

In this article, the phenomenon of libre software, especially the Bazaar-style development model, has been discussed from an software engineering perspective. The characteristics of this development model that has revolutionized the world of libre software and that has sparked an intense debate on the effectiveness and efficiency of its development process (in comparison to more traditional approaches) have been presented. On the other hand, it has been pointed out that we still have a very sparse knowledge of the Bazaar model. Among others, one of its shortcomings is that "success" is hardly predictable.

Libre software offers access to the source code and other elements that serve for the purpose of developer intercommunication in the software production process. Hence, they can be studied empirically. In this article we have seen three different perspectives: one closely bound to the analysis of source code, a second one dealing with the developers of libre software and, finally, one occupied with estimations of the production costs.

The above described studies of the source code archives demonstrate that a small development group is generally responsible for a big part of the code, whereas the comparably small number of external contributions comes from a greater amount of people. On the other hand, libre software is less inclined to contain errors due to its high release rate and fast feedback cycle. Also, some projects have a growth rate never seen before in similar-sized closed software projects. That can be explained partly through the possibilities for participation offered by the Bazaar model and partly because of an efficient technical and organizational modularization.

As regards the developers of libre software, this article has shown the importance of learning more about them since, from a management point of view, human resources are one of the pillars for cost estimation and prediction of software evolution. Libre software developers are a very heterogeneous group of people, and while the great majority dedicates only a few hours a week to the development of libre

[15] Cf. Antoniades et al. (2003).

software, a small but important minority dedicates almost as much time as a part-time job would require, or more. It is important to emphasize that in great projects that follow the Bazaar model, several generations of developers have been observed, continually following each other.

Predictions of cost and effort for libre software projects fail for two reasons. First, there is the lack of information on which we can count nowadays and secondly, because assumptions of traditional models cannot be applied. We have seen how some research projects attempted to do so and ended up without satisfying results.

Finally, some of the aspects have been analyzed that have to be treated in the next future. Of greatest importance is the creation of definitions so that semantic ambiguities disappear. It should be investigated further how to better characterize projects in order to extend the present knowledge and to be able to develop new methods and models. For that aim, the creation of a suite of tools that automates as far as possible the characterization of existing projects is proposed.

Besides such technical issues, there is a clear need to enhance studies of relationships among developers through more conventional social network analysis. The emergence of new forms of labor division, e.g. in the case of translators and designers deserves further attention.

Last, but not least, the possibility of having new methods of understanding libre software by means of intelligent agents is pointed out.

The reader who desires to obtain more information and data on the advances of software engineering in the field of libre software, may visit the Libre Software Engineering website at http://libresoft.dat.escet.urjc.es.

Bibliography

Antoniades, Ioannis, Samoladas Ioannis, Stamelos Ioannis, and Bleris G.L. (2003): *Dynamical simulation models of the Open Source development process*, pending publication in: Free/Open Source Software Development, edited by Stefan Koch, Idea Inc, Vienna.

Boehm, Barry W. (1981): *Software Engineering Economics*, Prentice Hall.

Campbell-Kelly, Martin (2003): *From Airline Reservations to Sonic the Hedgehog: a History of the Software Industry*, Cambridge, MA & London: MIT Press.

Capiluppi, Andrea, Patricia Lago, and Maurizio Morisio (2003): *Evidences in the evolution of OS projects through Changelog Analyses*, online http://softeng.polito.it/andrea/publications/icse2003.pdf (28.1.2004)

Challet, Damien and Yann Le Du (2003): *Closed source versus open source in a model of software bug dynamics*, (prepublished), online http://arxiv.org/abs/cond-mat/0306511 (28.1.2004)

Dempsey, Bart J., Debra Weiss, Paul Jones and Jane Greenberg (1999): *A Quantitative Profile of a Community of Open Source Linux Developers*, online http://www.ibiblio.org/osrt/develpro.html. (28.01.2004)

Ehrenkrantz, Justin R. (2003): *Release Management Within Open Source Projects*, online http://opensource.ucc.ie/icse2003/3rd-ws-on-oss-engineering.pdf (28.01.2004)

FLOSS (2002): Rishab Aiyer Ghosh, Rüdiger Glott, and Gregorio Robles, FLOSS: *Free/Libre and Open Source Software: Survey and Study*, online http://www.flossproject.org/ (28.01.2004)

Godfrey, Michael W. and Qiang Tu (2000): E*volution in Open Source Software: A Case Study*, online http://plg.uwaterloo.ca/~migod/papers/icsm00.pdf (28.01.2004)

González-Barahona, Jesús M., Miguel A. Ortuño Pérez, Pedro de las Heras Quirós, José Centeno González, and Vicente Matellán Olivera (2001): *Counting potatoes: The size of Debian 2.2*, in: Upgrade Magazine, Vol. 2, Issue 6, December 2001, online http://upgrade-cepis.org/issues/2001/6/up2-6gonzalez.pdf (28.01.2004)

González-Barahona, Jesús M. and Gregorio Robles (2003): *Unmounting the code god assumption*, proceedings of the Fourth International Conference on Extreme Programming and Agile Processes in Software Engineering (Genova, Italy), online http://libresoft.dat.escet.urjc.es/html/downloads/xp2003-barahona-robles.pdf (28.01.2004)

González-Barahona, Jesús M., et. al (2003b): *Anatomy of two GNU/Linux distributions*, pending publication in: Free/Open Source Software Development, edited by Stefan Koch, Idea Inc, Vienna.

Germán, Daniel and Audris Mockus (2003): *Automating the Measurement of Open Source Projects*, online http://opensource.ucc.ie/icse2003/3rd-ws-on-oss-engineering.pdf (28.01.2004)

Ghosh, Rishab Aiyer (1998): *Cooking-pot markets: an economic model for "free" resources on the Internet*, online http://www.firstmonday.dk/issues/issue3_3/ghosh/ (28.01.2004)

Ghosh, Rishab Aiyer and Vipul Ved Prakash (2000): *The Orbiten Free Software Survey*, online http://www.firstmonday.dk/issues/issue5_7/ghosh/index.html (28.01.2004)

Ghosh, Rishab Aiyer (2003): *Clustering and dependencies in free/open source software development: Methodology and tools*, online http://www.firstmonday.dk/issues/issue8_4/ghosh/index.html (28.01.2004)

Helay, Kieran and Alan Schussman (2003): *The Ecology of Open Source Software Development*, online http://opensource.mit.edu/papers/healyschussman.pdf (28.01.2004)

Koch, Stefan and Georg Schneider (2000): *Results from Software Engineering Research into Open Source Development Projects Using Public Data*, online http://wwwai.wu-wien.ac.at/~koch/forschung/sw-eng/wp22.pdf (28.01.2004)

Krishnamurthy, Sandeep (2002): *Cave or Community? An Empirical Examination of 100 Mature Open Source Projects*,
online http://opensource.mit.edu/papers/krishnamurthy.pdf (28.01.2004)

Lanzara, Giovan R. and Michelle Morner (2003): *The Knowledge Ecology of Open-Source Software Projects*,
online http://opensource.mit.edu/papers/lanzaramorner.pdf (28.01.2004)

Lehman, MM, J.F. Ramil, P.D. Wernick, and D.E. Perry (1997): *Metrics and laws of software evolution – the nineties view*, proceedings of the Fourth International Software Metrics Symposium.

Levy, Steven (2001): *Hackers: Heroes of the computer Revolution*, New York, Penguin Books.

MacCormack, Alan (2001): *How Internet Companies Build Software*, in: MIT Sloan Management Review, vol. 42, no. 2 (Winter 2001), p. 75–84.

Madey, Greg, V. Freeh, and R. Tynan (2002): *The Open Source Software Development Phenomenon: An Analysis Based on Social Network Theory*,
online http://www.nd.edu/~oss/papers/amcis_oss.pdf (28.01.2004)

Mockus, Audris, Roy T. Fielding, and James D. Herbsleb (2000): *Two Case Studies of Open Source Software Development: Apache and Mozilla*,
online http://www.research.avayalabs.com/techreport/alr-2002-003-paper.pdf (28.01.2004)

Park, Robert E. et. al. (1992): *Software Size Measurement: A Framework for counting Source Statements*, Technical Report CMU/SEI-92-TR-20,
online http://www.sei.cmu.edu/publications/documents/92.reports/92.tr.020.html (28.01.2004)

Raymond, Eric S. (1997): *The Cathedral and the Bazaar, Musings on Linux and Open Source by an Accidental Revolutionary*,
http://catb.org/~esr/writings/cathedral-bazaar/ (28.01.2004)

Robles, Gregorio, Jesús González-Barahona, José Centeno-González, Vicente Matellán-Olivera and Luis Rodero-Merino (2003): *Studying the evolution of libre software projects using publicly available data*, proceedings of the 3rd Workshop on Open Source Software Engineering held at the 25th International Conference on Software Engineering,
online http://opensource.ucc.ie/icse2003/3rd-ws-on-oss-engineering.pdf (28.01.2004)

Wheeler, David A. (2000): *Estimating Linux's Size*,
online http://www.dwheeler.com/sloc (28.01.2004)

Wheeler, David A. (2001): *More than a Gigabuck: Estimating GNU/Linux's Size*,
online http://www.dwheeler.com/sloc (28.01.2004)

WIDI (2001): Gregorio Robles, Henrik Scheider, Ingo Tretkowski, and Niels Weber, *WIDI – Who Is Doing It? A research on Libre Software developers*,
online http://widi.berlios.de/paper/study.pdf (28.01.2004)

Sicherheit mit Open Source – Die Debatte im Kontext, die Argumente auf dem Prüfstein

ROBERT A. GEHRING[1]

Einführung

Die Debatte darüber, welches denn nun der sicherere Weg der Softwareentwicklung sei, hält seit einigen Jahren an und wird wohl in absehbarer Zeit auch nicht beigelegt werden – dazu sind die auf dem Spiel stehenden Interessen einfach zu groß. Das Argument der Sicherheit hat in den vergangenen Jahren nicht zuletzt deswegen an Gewicht gewonnen, weil jährlich Milliardenschäden durch Viren, Würmer, Einbrüche in Computersysteme und, nicht zu vergessen, Softwarefehler vermeldet werden. Der Leidensdruck derjenigen, die auf verlässliche IT-Systeme[2] angewiesen sind, wächst stetig, und die Suche nach einem Ausweg gewinnt an Priorität.

Im vorliegenden Beitrag unternimmt der Autor den Versuch, die Debatte um die Sicherheit von Open-Source in ihren Grundzügen zu systematisieren. Ausgehend von der Identifizierung der tragenden Pfeiler der Debatte werden Defizite aufgezeigt und die Potentiale von Open-Source, die Sicherheit von IT-Systemen zu verbessern, insbesondere in den Bereichen Softwaretechnologie und Softwareökonomie diskutiert.

Der 1. Pfeiler: Sicherheit als Marketing-Argument

Open-Source-Software (OSS), so deren Protagonisten, biete einen Ausweg aus der andauernden Softwarekrise. Meinungsäußerungen wie, die Ergebnisse von Open-Source-Softwareentwicklung seien „[g]rundsätzlich besser als andere Software, weil eleganter und obendrein sicherer und zuverlässiger" (Dignatz 1999 unter Berufung auf Hal Varian) oder „Faster, Better, and Cheaper" (Scacchi 2002),[3] hört und liest der interessierte Beobachter immer aufs Neue.[4] Andere fragen eher skeptisch „Open Source Security: Opportunity or Oxymoron?" (Lawton 2002)

Man darf in diesem Zusammenhang jedenfalls nicht übersehen, dass aus dem, was 1984 als Richard Stallmans gegen die mit zunehmender Kommerzialisierung

---

[1] Wissenschaftlicher Mitarbeiter im Fachgebiet „Informatik und Gesellschaft" der TU Berlin. Der Autor dankt Dirk Kuhlmann und Matthias Bärwolff für ihre Kommentare zu diesem Aufsatz.

[2] „IT-Systeme" wird hier in einem generischen Sinne verwandt und beschreibt funktionale Konglomerate aus Hard- und Software, die in der Regel Netzwerkelemente zum Datenaustausch mit anderen IT-Systemen beinhalten.

[3] In der Fachzeitschrift Wirtschaftsinformatik gab es kürzlich einen Schwerpunkt zum Thema „Sicherheit durch Open Source" (Jg. 45, 2003, Nr. 4, S. 474–482, online: http://www.wirtschaftsinformatik.de/wi_heft.php?op=heft&heftid=140). Anhand der dort vorgestellten Beiträge kann man sich einen guten Überblick über das Meinungsspektrum verschaffen.

[4] Nicht immer stehen hinter solchen Meinungen um Objektivität bemühte Wissenschaftler. Man wird auch eine ganze Reihe tendenziell gegen Microsoft und (oder) sein Software-Monopol eingestellter Interessensgruppen zum Chor der „Open-Source ist sicherer"-Protagonisten zählen dürfen.

eingehenden „enclosures“[5] bei Software gerichtete Ein-Mann-Aktion des „last true hacker left on earth“ (Levy 2001, S. 427) begann, mittlerweile ein „Big Business“[6] geworden ist, in dem Milliardenumsätze getätigt werden.[7] Da „gehört Klappern zum Handwerk“, will man sich gegen die Konkurrenten aus dem Lager der Closed-Source-Software (CSS) durchsetzen; schließlich wird ja auch ein Open-Source-Antagonist wie Microsoft nicht müde, seine Direktoren und Verkäufer das Argument der Sicherheit bzw. Vertrauenswürdigkeit (*trustworthiness*, wie der neuere Microsoft-Terminus lautet) im Munde führen zu lassen.[8]

Der 2. Pfeiler: Sicherheit und Politik

Ein weiterer Faktor, der die Debatte um die Sicherheit von Open-Source befeuert, ist geostrategischer Natur. In der von US-Unternehmen dominierten PC-Softwarewelt der letzten 30 Jahre war es Unternehmen in anderen Ländern nur ausnahmsweise gelungen, sich zum „global player“ aufzuschwingen. SAP ist sicherlich das bekannteste Beispiel. In den meisten Fällen dominieren die von US-Unternehmen stammenden Softwareprodukte derartig den internationalen und lokalen Markt, dass ein echter Wettbewerb nicht in Gang kommen konnte.[9] Die rigorose Durchsetzung von Intellectual Property Rights (IPRs) amerikanischer Copyrightindustrien (inkl. Softwarehersteller) auf internationaler Ebene,[10] hat maßgeblich dazu beigetragen, die Positionen der US-Software-Industrie zu stärken. US-amerikanische Software dominiert in Indien, Japan, Korea, China, usw. die lokalen Märkte, und sei es in Form von illegalen Kopien. Nicht alle von diesen Ländern standen in einem freundschaftlichen Verhältnis zu den USA, sei es aus politischen (China) oder vor-

5 Der Terminus „enclosure“ beschreibt ursprünglich die Einhegung vormals frei zugänglichen Weidelandes („commons“), wie sie etwa seit dem 15. Jahrhundert in England stattfanden und unter dem Namen „fencing“ aus den USA des 19. Jahrhunderts bekannt sind. In den letzten Jahren wurde „enclosures“ von verschiedenen Autoren auch zunehmend auf Entwicklungen im Cyberspace angewandt.

6 So fragte ein Bericht der Deutsche Bank Research (Hofmann 2002). „One commentator has argued that ‚anarcho-communism is now sponsored by corporate capital.‘“ (Bollier 2003, S. 37, Fußnote ausgelassen). Dieser Umstand findet, wie Richard Stallman nicht müde wird, zu betonen, nicht seine ungeteilte Zustimmung. Siehe auch die Aussagen des GNU-Projekts in „Why ‚Free Software‘ is better than ‚Open Source‘“, http://www.gnu.org/philosophy/free-software-for-freedom.html (08. Jan 2004).

7 Laut Financial Times Deutschland vom 29.12.2003 wurde mit Linux-basierten Servern im Jahr 2002 ein Umsatz von 1,4 Mrd. US$ erwirtschaftet. Allein in den ersten drei Quartalen des Jahres 2003 stieg der Umsatz auf 2,2 Mrd. US$. IBM allein meldete für 2002 einen Umsatz im Linux-Geschäft von 1,5 Mrd. US$ und HP nannte 2 Mrd. US$. (Goltzsch 2003)

8 Sicherheit stünde für die nächsten „drei bis vier Jahre auf unserer Prioritätenliste ganz oben,“ äußerte sich Bill Gates (heise 2004). Im Streit um den Zusammenhang zwischen Offenheit des Quellcodes und die Sicherheit von Software beschwichtigt Microsoft: „Development models ultimately have a neutral effect on software security.“ (Mundie 2003)

9 Diese Dominanz ist, wie im Fall von Microsoft gerichtlich festgestellt wurde, zu einem gewichtigen Teil durch wettbewerbswidriges Verhalten zustandegekommen. Vgl. Ishii und Lutterbeck (2002).

10 Auf Drängen der Vertreter aus den „Copyright-Industrien“, Softwarehersteller eingeschlossen, wurde 1984 Abschnitt 301 des US-Trade Acts von 1974 dahingehend ergänzt, dass die mangelhafte Durchsetzung von IPRs in einem Land als „unfair trade barrier that could provoke U.S. retaliation“ (Ryan 1998, S. 11) betrachtet wurde. Bei den Verhandlungen der Uruguay-Runde zum Welthandelsabkommen GATT überraschte der US-Handelsbeauftrage 1986 mit dem Ansinnen, dass IPRs unbedingt in die Verhandlungen einzubeziehen seien (Ryan 1998, S. 1). Ein brasilianischer Beobachter

rangig aus wirtschaftlichen Gründen (etwa Indien und Japan). Man argwöhnte dort, dass US-Software auf einheimischen Computern nicht unbedingt vertrauenswürdig sei. Die Crypto-Debatte der 90er Jahre[11] um starke und schwache Verschlüsselung, Exportverbote,[12] Schlüsselhinterlegung und Hintertüren in Software, der Versuch der US-Regierung, mit dem Clipper-Chip eine definitiv abhörbare Technologie in den Markt zu drücken usw., stießen gerade auch bei Sicherheitsfachleuten auf scharfe Kritik, die zur damaligen Zeit in Kryptographie noch die Lösung für alle Sicherheitsprobleme im IT-Bereich sahen.[13] Auch die Versuche der US-Administration, in internationalen Verhandlungen Einschränkungen für den Einsatz von Verschlüsselungsverfahren durchzusetzen, trafen auf Missfallen, nicht zuletzt in Europa, wo man durchaus ernste Befürchtungen hinsichtlich High-Tech-Spionage hegte. Das Bekanntwerden des von den USA und einigen engen Verbündeten (Großbritannien, Australien, Neuseeland) betriebenen Abhörnetzwerkes Echelon[14] trug ein Übriges dazu bei, ein allgemeines Misstrauen gegenüber den USA zu stützen, das sich auch auf die von dort stammende Software erstreckte. Da man in den streng geheimgehaltenen Code ja nicht hineinsehen konnte, war man auch nicht in der Lage, dessen Vertrauenswürdigkeit zu beurteilen. Das war ein klares Argument *contra* Closed-Source-Software, das in der Folge zu einem Argument *pro* Open-Source-Software wurde: Bei offen gelegtem Quellcode kann sich im Prinzip jede/r davon überzeugen, ob die Software etwa auf der eigenen Festplatte spioniert und „nach Hause telefoniert." Dieses Argument spielte in der deutschen Debatte immer wieder eine große Rolle[15] und verfehlte auch anderswo seine Wirkung nicht. So haben diverse

schildert das Ergebnis so: „Industrialized countries forced developing countries to initiate negotiation of an agreement on TRIPS with the clear objective of universalizing the standards of IPRs protection that the former had incorporated in their legislation [...]" (Correa 2000). Die Vorgänge, die zur Verabschiedung von WIPO Copyright Treaty (WCT) und WIPO Performances and Phonograms Treaty (WPPT) im Dezember 1996 führten, zeigen eine ähnliche Entwicklungsgeschichte (Ryan 1998; Litman 2001; Drahos und Braithwaite 2002). Auch bei der Verabschiedung der gültigen Fassung der EU-Softwarerichtlinie (91/250/EWG) spielte die globale Auseinandersetzung in den „Softwars" (Clapes 1993) zwischen den USA und ihren von ihnen wahrgenommenen Wettbewerbern (EU, Japan) eine wesentliche Rolle.

11 Die Crypto-Debatte und ihre Vorgeschichte wird u.a. dargestellt in Diffie und Landau (1998).

12 Siehe etwa auch die Prozesse Bernstein v. United States Department of Justice, 176 F.3d 1132 (9th Cir. 1999) (Lemley, Menell, Merges und Samuelson 2002, S. 914–915) und Junger v. Daley, 209 F.3d 481 (6th Cir. 2000) (Maggs, Soma und Sprowl 2001, S. 819–823).

13 Der Irrtum, mit Kryptographie die Sicherheitsprobleme in vernetzten IT-Systemen lösen zu können, ist mittlerweile von vielen Fachleuten, nicht zuletzt von Bruce Schneier, erkannt worden: „In my vision cryptography was the great technological equalizer; anyone with a cheap (and getting cheaper every year) cornputer could have the same security as the largest government. In the second edition of the same book, written two years later, I went so far as to write: ‚It is insufficient to protect ourselves with laws; we need to protect ourselves with mathematics.' It's just not true. Cryptography can't do any of that. [...] Security, palpable security that you or I might find useful in our lives, involves people: things people know, relationships between people, people and how they relate to machines. Digital security involves computers: complex, unstable, buggy computers." (Schneier 2000, S. xi)

14 Vgl. Schulzki-Haddouti (2001).

15 Vgl. z. B. Köhntopp, Köhntopp und Pfitzmann (2000).

asiatische Regierungen ein verstärktes Engagement im Open-Source-Bereich entwickelt.[16]

Mit der zunehmenden Verfügbarkeit von Open-Source-Software in der ganzen Welt und ihrer wachsenden Leistungsfähigkeit, wird nicht nur die Abhängigkeit aller Länder von US-geliefertem Code reduziert, sondern es verliert das IPR-Paradigma hinter dem Closed-Source-Software-Modell selbst spürbar an Überzeugungskraft. Wie David Bollier es ausdrückt:

> [O]nline sharing and collaboration raise some profound questions about the foundations of intellectual property law, which assumes that useful creativity will emerge only through a sanctioned structure of markets, incentives, and contracts. (2003, S. 208)

Mit dem Einsatz von Open-Source-Software lösen die jeweiligen Länder nicht nur Ihr Sicherheitsdilemma (Vertrauensdilemma), sondern versetzen sich selbst in die Lage, eine eigene, konkurrenzfähige Softwareindustrie zu entwickeln. Die Reichweite der Verquickung des offensichtlichen Arguments (Sicherheit und Vertrauenswürdikeit) mit dem ökonomischen Argument (industrielle Eigenständigkeit) sollte in seiner Bedeutung nicht unterschätzt werden. Damit ordnet sich der Streit um die Sicherheit von Softwareentwicklungsmodellen ein in die weitaus größere Auseinandersetzung um die Frage, wie denn die künftige Verfasstheit der Informationsgesellschaft aussehen solle, und welche Rolle exklusive Eigentumsrechte dabei spielen werden[17] – nicht nur, aber insbesondere auch für Software.[18]

Der 3. Pfeiler: Sicherheit und Softwaretechnologie

Aus rein softwaretechnologischer Sicht wurden ebenfalls Argumente zugunsten eines Open-Source-Ansatzes vorgebracht. Das Open-Source-Entwicklungsmodell sei besser, so verlautet immer wieder, weil die so entstehende Software weniger Fehler aufweisen würde. „Given enough eyeballs, all bugs are shallow," formulierte Eric S. Raymond (1999, S. 41) in seinem berühmten Aufsatz „The Cathedral and the Bazaar" die These, dass die Offenlegung des Codes den Beta-Testern eine gründliche Inspektion im Sinne des wissenschaftlichen *peer reviews* ermöglichen würde.[19] Fehler

[16] Einem Artikel des Economist vom September 2003 zufolge „China has been working on a local version of Linux for years, on the grounds of national self-sufficiency, security and to avoid being too dependent on a single foreign supplier." In Korea ist die Regierung bemüht, Microsoft-Software durch Open-Source-Produkte zu ersetzen (Myung 2003). Und in Indien hat gar der Präsident persönlich dazu aufgerufen, Open-Source-Software zu entwickeln (Kalam 2003). Dass gerade Asien so offen ist für Open-Source, könnte vielleicht z.T. damit erklärt werden, dass dem Lernen durch Imitieren und Verbessern im Konfuzianismus (der in Asien über die die Jahrhunderte bekanntlich eine enorme Wirkung entfaltete) eine Schlüsselstellung zukommt (Wei-Ming 1993).

[17] Vgl. u.a. National Research Council (1991), Office of Technology Assessment (1992), Branscomb (1994), Lessig (1999), Imparato (1999), National Research Council (2000) und Bollier (2003) sowie Thierer und Wayne, Jr. (2003) (stellvertretend für die konservativen Auffassungen in den USA) und McChesney, Wood und Foster (1998) (stellvertretend für die Auffassungen der sozialistisch orientierten Kreise in den USA).

[18] Siehe auch Lutterbeck, Horns und Gehring (2000).

[19] Nach Mustonen (2003, S. 103) hat eine Reihe von Autoren argumentiert, dass neues Wissen in der arbeitsteiligen Gesellschaft typischerweise unter zwei unterschiedlichen Anreizstrukturen generiert wird. Neben der profitgesteuerten „technology" gibt es die Anreizstrukturen innerhalb von „science". Dort spielen „peer recognition" und die aus einer schnellen Veröffentlichung von neuem

würden daher sehr schnell entdeckt und beseitigt werden. Da viele Fehler aber gleichzeitig Sicherheitslücken darstellen würden, wären Open-Source-Programme sicherer, da sie weniger Fehler hätten. Eine solche Schlussfolgerung hat Raymond in seinem Aufsatz zwar selbst nicht unmittelbar gezogen, sie lag aber nahe,[20] ist der Zusammenhang zwischen bestimmten Fehlertypen und Sicherheitsmängeln durch langjährige Erfahrungen doch empirisch gut belegt. Immer wieder führen etwa so genannte Pufferüberläufe (buffer overflows), d.h. klassische Programmierfehler, zu verletzlichen Computersystemen, auf die erfolgreiche Angriffe möglich sind.[21] Zwar gibt es keine 1:1-Beziehung zwischen Softwarefehlern und Sicherheitslücken, aber beide Größen sind eng miteinander verknüpft,[22] was sich auch darin zeigt, dass in der Diskussion Begriffe wie „dependable", „robust", „trustworthy" und „reliable" alle mit einer Konnotation von Sicherheit daherkommen.[23] Andere Argumente im Sinne der Sicherheit durch Open-Source-Entwicklung sind die Möglichkeit, die Komplexität der Software zielgerichtet durch Entfernung von Code reduzieren zu können, sowie die durch die verteilte Entwicklung bedingte Modularität[24] des Codes. Die Stellungnahmen der meisten Fachleute zu diesen Argumenten bleiben allerdings insgesamt recht wage:

> Open source software can be potentially more secure than closed source software, in which source codes of programs can be examined by Peer Review from parties other than the creators. However the openness of source codes is not necessary an advantage by itself, the level of software security further depends on the efficiency of Peer Review,

Wissen zu erzielende Reputation die Rolle der maßgeblichen Stimulationsfaktoren: „Scientific recognition is achieved by making one's contribution public to peer review as quickly as possible and acquiring priority to the new knowledge." (Mustonen 2003, S. 103) Der Aspekt des *peer reviews* in der Open-Source-Entwicklung wurde genauer untersucht von Stark (2002), die eine positive Attitüde hinsichtlich des *peer reviews* (bei Open-Source-Entwicklern) nachgewiesen hat.

20 Sie wurde von verschiedenen Autoren im Laufe der letzten Jahre immer wieder gezogen, zuletzt von Oppliger (2003).

21 Zum Problem der Verletzlichkeit durch „buffer overflows" siehe z.B. Krsul, Spafford und Tripunitara (1998). Zur Bedeutung in der Praxis siehe Kamerling (2003), wo die 20 kritischsten Schwachpunkte für Windows- und UNIX-Betriebssysteme aufgelistet werden. Unter den kritischsten Schwachpunkten findet sich eine ganze Reihe, die auf „buffer overflows" zurückzuführen ist.

22 Eine relativierende Formulierung ist notwendig, da es keinen wissenschaftlich formulierten Zusammenhang zwischen Fehlern in und Sicherheit von Software, sondern lediglich Erfahrungswerte gibt. Sichere Software muss nicht fehlerfrei sein, fehlerfreie Software kann unsicher sein, und in den meisten Fällen wird man davon ausgehen müssen, dass die Software weder fehlerfrei noch sicher ist, wobei es in der Regel noch nicht einmal Einigkeit hinsichtlich der Terminologie gibt (Jonsson, Strömberg und Lindskog 2000). Das viel grundlegendere Problem aber ist, dass es bis dato keine wissenschaftlich fundierten Verfahren zur Entwicklung von sicherer Software gibt (Greenwals u.a. 2003), von fehlerfreier Software ganz zu schweigen. Der alternative Ansatz, Software-Code nicht zu schreiben, sondern nach formalen Methoden zu entwickeln, d.h. aus einer formal vollständigen Spezifikation zu generieren, hat seinerseits Tücken. Formale Methoden sind z.B. oft unökomisch (Luqi und Goguen 1997).

23 Vgl. u.a. Neumann (1998, S. 128): „‚[R]obust' is an intentionally inclusive term embracing meaningful security, reliability, availability, and system survivability, in the face of a wide and realistic range of potential adversities[...]"; Littlewood und Strigini (2000, S. 177): „Dependability encompasses, among other attributes, reliability, safety, security, and availability."; Knight (2002, S. 685) „The no-

the skill of reviewers and consistent support of software after the initial release." (Li 2002, S. 73)

Der 4. Pfeiler: Sicherheit und Software-Ökonomie

Software existiert nicht in einer idealen Welt, sondern wird von realen Menschen entwickelt und in einer realen Welt verteilt und eingesetzt. Die reale Welt ist eine Marktwirtschaft in der Software oft mit knappen Ressourcen produziert wird und als Ware den Bedingungen des Marktes genügen muss.[25] Dass das Sicherheitsargument regelmäßig dazu dient, Open-Source-Software zu vermarkten, mag als (schwacher) Beleg dafür gelten.

Der Zusammenhang zwischen Softwarequalität und Softwareökonomie im Allgemeinen wird, im Gegensatz zum technologischen Zusammenhang, erst wieder in jüngerer Zeit diskutiert und untersucht.[26] Ausgangspunkt ist dabei üblicherweise eine mikroökonomisch fundierte Kosten-Nutzen-Betrachtung: Es erfordert nicht unerhebliche Investitionen, Systeme sicherer zu machen, d.h. das mit ihrem Einsatz verbundene Risiko zu senken. Wenn die notwendigen Investitionen höher ausfallen als das in einem möglichen Schadensfall realisierte Risiko, ist es ökonomisch sinnvoller, die Investitionen nicht zu tätigen: „An economics perspective naturally recognizes that while some investment in information security is good, more security is not always worth the cost." (Gordon und Loeb 2002) Stattdessen könnte etwa ein Teil davon in die Absicherung von anderen Risiken mit höherem Schadenspotential fließen.[27]

Weitgehend unberücksichtigt blieben bisher Externalitäten, d.h. das Problem, dass das Schadensrisiko nicht notwendig bei demjenigen eintritt, der sein System nicht absichert. Man denke etwa an Internetserver die „gehackt" und als Viren- bzw. Spamverteiler genutzt werden.[28] In diesem Falle entsteht der überwiegende

tion of dependability is broken down into six fundamental properties: (1) reliability; (2) availability; (3) safety; (4) confidentiality; (5) integrity; and (6) maintainability. [...] It is important to note that security [...] is not included explicitly in this list. Rather, it is covered by the fourth and fifth items in the list.". Siehe weiterhin Anderson (2001), Viega und McGraw (2002), Meadows und McLean (1999) und Parnas, van Schouwen und Kwan (1990).

24 Den Zusammenhang zwischen Modularität, Code-Umfang und Softwarequalität diskutieren z.B. Stamelos, Angelis, Oikonomou und Bleris (2002).

25 Zu der Frage, inwiefern auch Open-Source-Software i.A. als Ware betrachtet werden muss, siehe den Beitrag von *Heller* und *Nuss* im vorliegenden Buch.

26 Vgl. u. a. Anderson (2001a), Gordon und Loeb (2002), Hoglund (2002), Sandhu (2003) sowie schwerpunktmäßig WEIS (2002; 2003). Bereits in Boehm (1981) wurde der Zusammenhang diskutiert, geriet zwischendurch aber – trotz gelegentlicher Publikationen zum Thema (z.B. Levy 1987) – weitgehend in Vergessenheit, wie auch beklagt wurde: Die Ökonomie sei „A Missing Link in Software Engineering," so der Titel eines Beitrages von Tockey (1997).

27 Die Frage der richtigen Investitionen in IT-Sicherheit wurde bis dato noch nicht zufriedenstellend beantwortet und es gibt diesbezüglich einige bemerkenswerte Widersprüche. Am 13. Februar 2002, zum Beispiel, legte das Office of Management and Budget dem US-Kongress einen Bericht vor, in dem die Ausgaben der US-Regierung für IT-Sicherheit evaluiert wurden. Der dortige Befund ist bemerkenswert: „Among other findings, the report stated they found no evidence of correlation between the amount spent on infomration security and the level of security performance." (Carini und Morel 2002).

28 Zum Zeitpunkt der Fertigstellung dieses Aufsatzes kursieren im Internet gerade zwei Varianten eines Wurms namens „MyDoom", deren Ausbreitungsgeschwindigkeit und -reichtweite alles bisher

Schaden gar nicht beim Serverbetreiber, sodass dessen Anreize, in die Serversicherheit zu investieren begrenzt sind – zumal er für den so angerichteten Schaden in der Regel auch nicht haftbar gemacht werden kann. Zwar wurde zur Lösung dieses Problems gelegentlich Versicherungsmodelle[29] oder (neue) Haftungsregelungen[30] ins Spiel gebracht, ohne dabei allerdings die Besonderheiten des Haftungsrechts hinsichtlich des Schadensnachweises und der dafür notwendigen Beweisführung in Sachen Ursächlichkeit zu berücksichtigen.

Ein Vorschlag der seit Jahren immer wieder einmal auftaucht ist, Softwareentwickler zu lizenzieren,[31] um so zu erreichen, dass nur „gute" Softwareentwickler Software schreiben, deren durchschnittliche Qualität dann natürlich steigen würde. Allerdings hält sich die Begeisterung für diesen Vorschlag in Grenzen und die Resultate des Open-Source-Modells scheinen seiner Sinnfälligkeit auch zu widersprechen.

Nur wenige Autoren haben sich bereits mit Fragen des Zusammenhangs zwischen den Spezifika der Software-Ökonomie (als konkreter Anwendung der Informationsökonomie) und ihren Auswirkungen auf die Sicherheit von OSS befasst. Zu nennen sind in dieser Hinsicht Landwehr (2002), Gehring (2003) sowie Franke und von Hippel (2003).[32] Der Tenor aller genannter Beiträge lautet dahingehend, dass die Open-Source-Methode mit ihren charakteristischen Eigenschaften,

- den Informationsfluss zwischen Programmierern und Anwendern zu verbessern, und
- den Anwendern die (rechtlichen und technischen) Mittel an die Hand zu geben, das System in seiner Substanz, im Quellcode, zu ändern,

einen positiven Beitrag zur Sicherheit leistet. Diesen Auffassungen zu Grunde liegt die Feststellung, dass dem proprietären Modell der Softwareentwicklung und -vermarktung Schwächen inhärent sind,[33] die hinsichtlich der Produktqualität unter dem Blickwinkel von Sicherheit wohl nicht innerhalb dieses Modells überwunden werden können. Open-Source bildet dann einen Ausweg aus der Perspektive des Anwenders.

Dagewesene in den Schatten stellt. Der Wurm kommt als Anhang einer E-Mail mit gefälschtem Absender und irreführender Betreffzeile und erweckt den Anschein einer fehlgeleiteten E-Mail. Nach Öffnen des Anhangs auf einem Microsoft-Betriebssystem installiert der Wurm eine Hintertür im PC und bereitet diesen darauf vor, in einem konzertierten Angriff Anfang Februar die Webserver von SCO und Microsoft lahmzulegen. Die infizierten Rechner selbst werden vorerst nicht in Mitleidenschaft gezogen, allerdings verheisst die Hintertür für die Zukunft nichts Gutes.

29 Für eine Versicherungslösung als „A Market Solution to the Internet Security Market Failure" plädieren z.B. Yurcik und Doss (2002).

30 Für eine Haftungslösung sprechen sich z.B. Fisk (2002) und Schneier (2002) aus; Martin (2002) äußert sich skeptisch hinsichtlich der Haftungsvorschläge.

31 Dafür z.B. Dorchester (1999), Frailey (1999), Tripp (2002); dagegen z.B. Kolawa (2002).

32 Anderson (2003) spricht das Thema zwar auch – aus theroretischer Perspektive – an, geht aber nicht weiter auf die Praxis ein und fordert statt dessen mehr empirische Untersuchungen. Ähnliches gilt für Lutterbeck, Horns und Gehring (2000).

33 Das betrifft etwa das Missverhalten, neue Produkte auf den Markt zu bringen (um neue Einnahmen zu generieren), ohne die Fehler in bereits vermarkteten Produkten vorher zu beseitigen.

Kritik: Defizite der Debatte

Weitgehend übersehen[34] wurde bisher ein weiteres Argument, jedenfalls insofern es um den Beitrag zu Softwarequalität und -sicherheit geht: der Faktor Mensch oder, mit anderen Worten, Psychologie und Soziologie[35] der arbeitsteiligen Softwareentwicklung. So spielt das Klima in der Projektorganisation eine wesentliche Rolle bei der Aufdeckung und Weitermeldung von Problemen während der Durchführung von Softwareprojekten (Tan, Smith, Keil und Montalegre 2003).

Bereits zu Beginn der 1970er Jahre sind die kognitiven Grenzen des individuellen Softwareentwicklers und die daraus zu ziehenden Schlussfolgerungen für die Arbeitsteilung bei der Softwareentwicklung betont worden. In seinem zeitlosen Standardwerk von 1971 beschreibt Gerald M. Weinberg „Programming as a Social Activity."[36] Komplexe Programme werden, Ausnahmen mögen die Regel bestätigen, in einem Prozess sozialer Interaktion entwickelt, betont Weinberg (1971, S. 52).[37] Als grundsätzlich schädlich betrachtet er hingegen – aus softwaretechnischer Perspektive – das Besitzstandsdenken bei Software:

> Well, what is wrong with ‚owning' programs? Artists ‚own' paintings; authors ‚own' books; architects ‚own' buildings. Don't these attributions lead to admiration and emulation of good workers by lesser ones? (S. 53)

Weinberg verneint die suggestiv naheliegende Schlussfolgerung und weist darauf hin, dass „Fehler" in klassischen Kunstwerken allenfalls eine Geschmackssache seien. Bei Software hingegen entscheidet letzten Endes der Computer, was ein Fehler ist – durch Fehlfunktionen des Programmes. Wo ein Künstler das Urteil seiner subjektiven Kritiker im Prinzip nach Belieben verwerfen kann, ist der Programmierer mit einer „objektiven Kritik" konfrontiert, aus deren Urteil er, wenn ein Fehler auftrit, im Grunde nur eine Konsequenz ziehen kann:

> This program is defective. This program is part of me, an extension of myself, even carrying my name. I am defective. (S. 54)

---

34 Die Ausnahme bildet eine kurzen Erwähnung Weinbergs und des „egoless programming" (s.u.) bei Raymond (1999, S. 61).

35 Zwar gibt es mittlerweile eine ganze Reihe soziologischer Untersuchungen zu Open-Source-Prozessen, vgl. etwa Robles, Scheider, Tretkowski und Weber (2001), FLOSS (2002) und Hertel, Niedner und Herrmann (2003), doch konzentrieren diese sich vorrangig auf Fragen von Herkunft, Motivation, Qualifikation usw. Die dabei gewonnenen Erkenntnisse sind hinsichtlich der Einflüsse der Soziologie auf Softwarequalität und -sicherheit wenig aussagekräftig, bzw. wurden dahingehend nicht weiter ausgewertet. Zum Einfluss des Betriebsklimas auf Softwareprojekte siehe z.B. Tan, Smith, Keil und Montalegre (2003).

36 So der Titel von Teil 2 in Weinberg (1971).

37 Diese unter dem Paradigma des Software-Engineering ein wenig in Vergessenheit geratene Einsicht ist in jüngerer Zeit zu neuen Ehren gekommen, z.B. in Hohmann (1996). Unter dem Schlagwort vom „agile programming" wird die Bedeutung der sozialen Interaktion für eine erfolgreiche Projektdurchführung mit Emphase betont. Vgl. z.B. Beck (2000) und Cockburn (2002).

Diese Erfahrung stellt den Programmierer vor ein Dilemma kognitiver Dissonanz[38]: Fehler in Programmen verletzen das Selbstwertgefühl des Entwicklers – mit Folgen für die Qualitätssicherung:

> A programmer who truly sees his program as an extension of his own ego is not going to be trying to find all the errors in that program. On the contrary, he is going to be trying to prove that the program is correct – even if this means the oversight of errors which are monstrous to another eye. All programmers are familiar with the symptoms of this dissonance resolution – in others, of course. (S. 55)

Diese Erkenntnisse wurden von Praktikern, wie etwa dem IBM-Forscher Glenford J. Myers (1976; 1979), bestätigt und in Konzeptionen für das erfolgreiche Entwickeln und Testen von Software (mit dem Ziel, die „software reliability" zu steigern) berücksichtigt. Testverfahren wurden so gestaltet, dass sie dem von Weinberg propagierten Ideal des „egoless programming"[39] (1971, S. 56) nahe kamen, das Myers (1976, S. 143) so beschreibt:

> Rather than being secretive and defensive toward his program, the programmer adopts the opposite attitude: his code is really a product of the entire project and he openly asks other programmers to read his code and constructively criticize it. When someone finds an error in his code, the programmer of course should not feel good about making a mistake, but his attitude is, "Wow, we found an error in our product! Good thing we found it now and not later! Let's learn from this mistake and also see if we can find another." A programmer who finds an error in another programmer's code does not respond with, "Look at your stupid mistake"; instead the response is more like "How interesting! I wonder if I made the same mistake in the module I wrote." Notice the fine point concerning cognitive dissonance in the previous sentence: I used the phrase "the module I wrote" instead of "my module".

Wohl nicht zufällig sind die Gemeinsamkeiten mit dem wissenschaftlichen *peer review*.[40] Man erkennt in der Beschreibung von Myers ohne weiteres dieselbe Attitüde wieder, Code nicht als Eigentum, sondern als etwas Gemeinsames, als Extension eines „commons" zu begreifen, wie sie aus den Selbstbeschreibungen der Open-Source-Community hinlänglich bekannt ist. Folgt man Weinberg (1971), Myers

---

38 Unter kognitiver Dissonanz versteht man die „Unvereinbarkeit von mehreren Überzeugungen, Einstellungen, Haltungen gegenüber Umweltsituationen, anderen Menschen u. deren Anschauungen, den eigenen Verhaltensnormen oder Wertmaßstäben u.a." (Hillmann 1994, S. 419). Eine Situation, die kongitive Dissonanz auslöst, wird als bestrafend empfunden, was Versuche des Ausweichens und Verdrängens nach sich zieht. (a.a.O.) Wenn also, begünstigt durch die propagierten Wertauffassungen und das soziale Umfeld, Fehler in Programmen als Bestrafung des eigenen Verhaltens (des Programmierers) angesehen werden oder nach sich ziehen, wird der Programmierer das Bekanntwerden von Fehlern vermeiden wollen, sogar sich selbst gegenüber (indem sie „übersehen" werden): „Programmers, if left to their own devices, will ignore the most glaring errors in their output – errors that anyone else can see in an instant." (Weinberg 1971, S. 56)

39 Weinberg (1971, S. 56) führt einen der Urväter des modernen PCs, John von Neumann, als Kronzeugen für die Vorteile des „egoless programming" an: „John von Neumann himself was perhaps the first programmer to recognize his inadequacies with respect to examination of his own work. Those who knew him have said that he was constantly asserting what a lousy programmer he was, and that he incessantly pushed his programs on other people to read for errors and clumsiness."

40 Zur Bedeutung des *peer review* in der Wissenschaft vgl. z.B. Macrina (1995, S. 69–95).

(1976; 1979) und anderen, so lassen sich Einsichten in den Zusammenhang zwischen Softwareentwicklungsprozess und Produktqualität gewinnen, die eine Fokussierung auf Software als „geistiges Eigentum“ (gemäß dem herrschenden Autorenparadigma[41]) aus softwaretechnischer Perspektive grundsätzlich in Frage stellen könnten.[42] Software ist eben kein Buch, das man sich ins Regal stellt, auch wenn es im Wortlaut des deutschen Urheberrechts bloß als ein Sprachwerk unter anderen, wie „Schriftwerke“ und „Reden“, eingestuft wird (UrhG §2 Abs. 1 Zif. 1).

Diese Ausführungen zu Soziologie und Psychologie der Softwareentwicklung sollen soweit genügen, um eines zu zeigen: In der Debatte um Open Source vs. Closed Source gibt es noch eine ganze Reihe nicht ausgeleuchteter Gassen und Winkel die näherer Untersuchungen wert sind.

Summa: Die Debatte um Open-Source und Sicherheit

Fassen wir einmal kurz zusammen. Vier Pfeiler tragen bisher die Debatte um die Sicherheit von Open-Source-Software:

1. Sicherheit als Verkaufsargument;
2. Zugang zum Quellcode als Voraussetzung für Sicherheit und Vertrauenswürdigkeit – das politische Argument;[43]
3. Open-Source-Entwicklungsmethodik als weniger fehlerträchtig – das technologische Argument;
4. Open-Source als alternativer Ansatz zur Softwareproduktion und -distribution – das ökonomische Argument.

Wer hat nun Recht, wenn es um die Sicherheit von Open-Source geht? Welche Argumente sind stichhaltig(er)?

---

41 Zur Problematik des Autorenparadigmas an und für sich vgl. Boyle (1996).

42 In den Anfangszeiten, als im Hinblick auf Software von geistigem Eigentum noch nicht die Rede war, gelangten einige der ersten Softwareentwickler überhaupt, die Wissenschaftler des EDSAC-Projekts an der Universität Cambridge in England, nach den ersten Erfahrungen mit Bugs (d.h. Fehlern in Programmen) und Debugging (d.h. dem Prozess der Fehlerbeseitigung) zu einer mit dem Autorenparadigma (der IPR-Regime) ziemlich unvereinbaren Auffassung:
„The Cambridge Group decided that the best way of reducing errors in programs would be to develop a ‚subroutine library‘ – an idea that had already been proposed by von Neumann and Goldstine. It was realized that many operations were common to different programs – for example, calculating a square root or a trigonometrical function, or printing out a number in a particular format. The idea of a subroutine library was to write these operations as kind of miniprograms that programmers could then copy into their own programs. In this way a typical program would consist of perhaps two-thirds subroutines and one-third new code.“ (Campbell-Kelly und Aspray 1996, S. 185 f.)
In diesem Sinne sollte Programmieren also überwiegend („two-thirds“) aus Kopieren bestehen, um die Qualität zu sichern. Jedes Programm wäre somit weniger Ausdruck der Individualität seines Entwicklers als vielmehr Resultat des Gebrauchs sozial hergestellter Erkenntnisse und Artefakte.

43 Im Zusammenhang mit elektronischen Wahlsystemen hat dieses Argument in jüngster Zeit, insbesondere nach Bekanntwerden eklatanter Schwächen in proprietären Closed-Source-Systemen, enorm an Bedeutung gewonnen. Vgl. z.B. Zetter (2004). Manche Fachleute sind allerdings der Meinung, dass softwarebasierte Wahlsysteme grundsätzlich als unsicher zu erachten seien. Entsprechend äußert sich z.B. David Dill: „As a computer scientist, I don‘t think they can make it secure enough, no matter what their procedure, or how they design the machine, or how the machines are inspected at independent laboratories.“ (Jonietz 2004, S. 74).

Aus praktischen Gründen, ist es an dieser Stelle nicht möglich, die gesamte verfügbare Literatur auszuwerten, wirklich alle Argumente abzuwägen. Stattdessen wird die weitere Untersuchung im Anschluss auf zwei der genannten Pfeiler fokussiert: Entwicklungsmethodik und Ökonomie. Deren Argumente werden im folgenden im Detail vorgestellt und anhand der verfügbaren Forschungsliteratur diskutiert.

Sicherheit und Softwaretechnologie

Im Laufe der letzten Jahre ist in Fachbüchern zur Software-/IT-Sicherheit das eine oder andere Mal auch Open-Source angerissen worden,[44] allerdings eher anekdotisch. Systematische Untersuchungen und theoretische Analysen sind noch rar gesät. Aussagekräftig erscheinen vor allem drei Aufsätze aus jüngerer Zeit. McCormack (2001) und Payne (2002) wenden sich der Frage von Qualität resp. Sicherheit von Open-Source empirisch zu, während Challet und Le Du (2003) ein theoretisches Modell zur Fehlerdynamik in der Closed-Source- und Open-Source-Entwicklungsmethode vorstellen. Die Erfahrungen, Argumente und Schlussfolgerungen der Autoren werden hier in aller Kürze referiert.

Alan McCormack hat sich zusammen mit Marco Iansiti und Roberto Verganti zwei Jahre lang in einer empirischen Feldstudie mit den Erfolgen oder Misserfolgen von Softwareprojekten und den unterschiedlichen Ansätze zu ihrer Durchführung befasst (McCormack 2001). Dabei lag der Fokus nicht auf der Frage der Code-Offenheit oder -Geschlossenheit, sondern darauf, ob klassische Entwicklungsmodelle (à la Wasserfall) oder modernere, evolutionäre Modelle erfolgversprechender sind.[45] Dazu wurden 29 abgeschlossene Softwareprojekte (durchgeführt von 17 Firmen) analysiert und die Faktoren herausgearbeitet, die am meisten zu einem erfolgreichen Projektabschluss[46] beigetragen haben (McCormack 2001, S 79). Vier kritische Erfolgsfaktoren konnten identifiziert werden:

- „Early Release of the Evolving Product Design to Customers;“ (S. 79)
- „Daily Incorporation of New Software Code and Rapid Feedback on Design Changes;“ (S. 81)

---

44 Vgl. Schneier (2000, S. 363–365), Viega und McGraw (2002, S. 75–89) und Anderson (2001, S. 536–537).

45 Zwar sind evolutionäre Entwicklungsmodelle in der Softwaretechnik seit Jahrzehnten bekannt, doch werden die meisten Projekte nicht nach dieser Methode entwickelt. Ein schwerwiegender Grund dafür dürfte die Organisation der Software-Entwicklung gemäß einem Prinzipal-Agent-Schema sein: Es gibt eine klare, marktbedingte Trennung zwischen Auftraggeber/Konsument und Hersteller/Lieferant einer Software. Damit verbunden sind die üblichen Probleme des Prinzipal-Agent-Schemas, die aus den mit einer solchen Trennung unvermeidlich einhergehenden Informationsasymmetrien resultieren. Grundlegende Darstellungen der Theorie findet sich in Laffont und Martimort (2002) und Macho-Stadler und Pérez-Castrillo (2001). Eine Anwendung der Theorie auf Software und die Konsequenzen für die Sicherheit findet sich in Gehring (2003). Siehe auch den Abschnitt Sicherheit und Software-Ökonomie des vorliegenden Aufsatzes, der die Argumente noch einmal zusammenfasst.

46 Maßgeblich für die Beurteilung des Projekterfolges waren die Einschätzungen einer Runde von 14 unabhängigen Industriexperten, die in einem 2-Runden-Delphi-Verfahren über Parameter wie „reliability, technical performance (such as speed) and breadth of functionality“ zu befinden hatten (McCormack 2001, S. 79).

- „A Team With Broad-Based Experience of Shipping Multiple Projects;" (S. 81)
- „Major Investments in the Design of the Product Architecture." (S. 82)

Das Ergebnis der Untersuchung lässt keinen Zweifel an der Überlegenheit evolutionärer Entwicklungsmethoden stehen:

> Successful development was evolutionary in nature. Companies first would release a low-functionality version of a product to selected customers at a very early stage of development. Thereafter work would proceed in an iterative fashion, with the design allowed to evolve in response to the customers' feedback. (McCormack 2001, S. 76)

Je weniger Funktionalität im ersten Release der Software, das den Testern zur Verfügung stand, enthalten war, desto höher fiel die Qualität des Endproduktes aus (McCormack 2001, S. 80). Ausschlaggebend ist nach McCormack beim „feedback" nicht die Anzahl der Tester (McCormack 2001, S. 80), sondern die Geschwindigkeit mit der „feedback" auf Änderungen in der Software erfolgt (McCormack 2001, S. 81). Was Raymond als wesentliches Erfolgskriterium der Open-Source-Debugging-Methode beschreibt, scheint tatsächlich als qualitätsrelevantes Paradigma gültig zu sein: „Release early. Release often. And listen to your customers!" (1999, S. 39)

McCormacks Befunde unterstreichen das und er selbst scheint das direkt zu bestätigen, indem er den Linux-Kernel als ein erfolgreiches evolutionäres Projekt anführt (2001, S. 83).

Die Untersuchung von Payne (2002) konzentrierte sich darauf herauszufinden, inwiefern der Open-Source-Prozess geeignet sei, die Sicherheit von Betriebssystemen zu gewährleisten. Faktoren die nach Payne (2002) dafür sprechen könnten sind „[p]eer review" (S. 63), „[f]lexibility and freedom" (S. 65) und andere „miscellaneous arguments" (S. 66). Nach einer kritischen Würdigung auch der Argumente contra Sicherheit bei Open-Source, werden die Ergebnisse einer Studie von 1999 vorgestellt, die den Entstehungsprozess dreier UNIX-artiger Betriebssysteme untersucht hat: Solaris als proprietäres System, Debian GNU/Linux und OpenBSD als Open-Source-Systeme (S. 72). Im Mittelpunkt der Beobachtungen standen die vier „security ‚dimensions': confidentiality, integrity, availability and audit" (S. 72). Denen wurde nach einer in einem früheren Aufsatz (Payne 1999) erläuterten Metrik numerische Werte zugeordnet, die eine Vergleichbarkeit der einzelnen Systemeigenschaften gewährleisten. So kommen die Betriebssysteme hinsichtlich der oben auf-

geführten Sicherheitsdimensionen bei den Sicherheitseigenschaften[47] („features"; S. 73) auf folgende Werte (größere Zahlen stehen für bessere Werte):

- Debian: 6,42; Solaris: 5,92; OpenBSD: 7,03

Man erkennt einen leichten Vorteil beim Open-Source-System Debian ggü. dem proprietären Solaris und einen deutlicheren Vorsprung von OpenBSD ggü. beiden Systemen. Diesen theoretisch ermittelten Werten wurden dann die tatsächlich festgestellten Schwachstellen („vulnerabilities"; S. 73) zur Seite gestellt und analog bewertet. Folgende Werte wurden dabei ermittelt (kleinere Zahlen stehen für geringere Angreifbarkeit/höhere Sicherheit):

- Debian: 7,72; Solaris: 7,74; OpenBSD: 4,19

Während Debian und Solaris hinsichtlich ihrer Schwachstellen praktisch gleichauf liegen, weist OpenBSD deutlich weniger Schwachpunkte auf. Payne (2002) setzt die Werte aus „features" und „vulnerabilites" dann für jedes Betriebssystem zueinander ins Verhältnis und ermittelt einen „Final Score" (S. 73) für jedes System (größere Zahlen stehen für eine bessere Bewertung):

- Debian: -1,0; Solaris: -3,5; OpenBSD: 10,2

Die beiden Open-Source-Betriebssysteme schneiden besser als das proprietäre Solaris ab, wobei der Vorsprung von Debian nicht wirklich überzeugend ausfällt. Nicht zu übersehen ist jedoch der Unterschied zwischen Solaris/Debian auf der einen und OpenBSD auf der anderen Seite. Der Autor schlussfolgert zu Recht, dass „the significant differences between Debian and OpenBSD's score support the argument that making a program ‚open source' does not, by itself, automatically improve the security of the program" (S. 76). Was dann, fragt der Autor weiter, erklärt den großen Unterschied zwischen Debian und OpenBSD? Payne beantwortet die Frage folgendermaßen:

> [W]hile the source code for the Debian system is available for anyone who cares to examine it, the OpenBSD source code is regularly and purposefully examined with the explicit intention of finding and fixing security holes (Payne, 1999), (Payne, 2000). Thus it is this auditing work, rather than simply the general availability of source code, that is responsible for OpenBSD's low number of security problems. This point bears repeating: software will not become automatically more secure by virtue of its source code being published. (2002, S. 76)

Zwar bietet der offen liegende Quellcode bei Open-Source-Software die Voraussetzung, um Software sicherer zu machen, aber die Gestaltung des Entwicklungsprozesses ist entscheidend für die Erfüllung der Erwartungen! Nur, wenn von

[47] Payne (2002, S. 72) erläutert, wie dabei vorgegangen wurde: „Confidentiality features studied included mechanisms such as cryptographic file systems or other data encryption systems. Some of the operating systems considered included the ability to prevent processes from modifying files in certain ways, even if the process were executing with superuser privileges and this acted as an integrity-based security feature. The systems also included security controls specifically relating to the availability dimension which allowed the amount of resources utilized by each process to be specified and limited. The objective here is to prevent a malicious user from over-utilizing a shared resource and denying other users access. Different logging mechanisms were often included such as special software to log network packets received by the computer or to log information about the system call requests made by processes to the kernel. These are examples of some of the availability features included in the systems studied."

der Möglichkeit des Auditing in der Praxis auch Gebrauch gemacht wird, kann der Code sicherer gemacht werden. Offener Code allein ist keine hinreichende Bedingung, wenn auch eine erfolgversprechende Voraussetzung.

Jedwede Diskussion um das Pro und Contra der Sicherheit von Software müsste in diesem Sinne weniger auf das Produkt fokussiert werden als auf den Prozess seiner Herstellung und Distribution. Der „magische Blick" in den Quellcode ist allein kein Garant für einen qualitäts- und sicherheitsbewussten Prozess.[48]

Den Entwicklungsprozess von Software haben Challet und Le Du (2003) in einem theoretischen Modell zur Beschreibung der Fehlerdynamik beschrieben.[49] Ziel ihrer Modellierung war es, unter Annahme typischer Randbedingungen in offenen und geschlossenen Softwareentwicklungsprozessen (Programmierer machen immer Fehler; es gibt nur wenige „geniale" Programmierer, die meisten sind nur mittelmäßig gut; unterschiedliche Programmierer steuern unterschiedlich viel Code zu einem Projekt bei; neuer Code bringt neue Fehler ein; Anwender können potentiell Fehler an die Entwickler melden; der Betreuer eines Projekts entscheidet, welcher Code neu integriert wird; Anwender modifizieren, bei Open-Source, den Code z.T. selbst, um Fehler zu beseitigen) die Entwicklung der Fehlerdynamik, d.h. die phasenweise Zu- und Abnahme von Fehlern im Projektcode, abzubilden. Die Richtigkeit ihres Modells zu Grundegelegt,[50] kommen sie hinsichtlich Closed-Source-Software zu folgender Einsicht:

> This figure already shows that closed source projects are always slower to converge to a bug-free state than open source projects at constant parameters. In addition, ignoring bug reports on already modified code is the best option for closed source projects; this even outperforms open source at short time scales, because the programmers only work on fully buggy parts, hence the bug fixing rate is higher. (S. 3)

---

48 Daher greift auch jede Kritik an Open Source zu kurz, die sich im Kern mit dem „magischen Blick" auseinandersetzt – wie auch jede Befürwortung mit diesem Argument. Insofern ist Oppliger (2003) zuzustimmen. Inwiefern allerdings die von ihm angeführten Randbedingungen zur positiven Beeinflussung der Sicherheit von Software, nämlich (1) ein vollständiger, konsistenter Entwurf; (2) gut (in Sicherheit) geschulte Softwareentwickler; (3) qualitativ hochwertiger Software zur Erstellung; (4) umfassende Tests der erstellten Software; (5) besondere Beachtung der Benutzerschnittstelle; und (6) professionelles, institutionalisierten Patch-Management notwendig seien (S. 675), tatsächlich in ihrer Gesamtheit zutreffen, wird nicht nachgewiesen. Die Arbeiten anderer Autoren lassen jedenfalls Zweifel zu. Die Erkenntnisse von McCormack (2001) etwa sprechen zumindest gegen (1). Und ob bei den am weitesten verbreiteten proprietären Systemen von professionellem Patch-Management gesprochen werden kann, wird jeder mit Skepsis beurteilen, der einmal versucht hat, einen Microsoft-Server auf dem aktuellen Stand zu halten, vgl. z.B. Forno (2003). Jedenfalls lassen sich die von Oppliger angeführten Kriterien nach Auffassung des Autors des vorliegenden Beitrages nicht so zusammfassen, dass „die Diskussionen um Open-Source-Software und entsprechende Lizenzierungsmodelle [...] nicht (sinnvoll) geführt werden können" (Oppliger 2003, S. 675). Zumal wenn man berücksichtigt, dass viele der hier angeführten Quellen, z.B. McCormack (2001) oder Franke und von Hippel (2003), nicht erkenntlich ausgewertet wurden und Fragen der Softwareökonomie weitestgehend unbeachtet blieben.

49 Zur Praxis der Qualitätssicherung in Open-Source-Projekten siehe Zao und Elbaum (2003).

50 Der Aufsatz von Challet und Le Du (2003) wurde zum *peer review* eingereicht, die Aussagen hier stammen alle aus der Vorabversion, wie im Literaturverzeichnis angegeben.

Stimmt diese aus dem theoretischen Modell gewonnene Erkenntnis, könnte sie erklären, warum, bei den kurzen Produktzyklen, die PC-Software heute aufweist, die Hersteller so wenig Energie in die Beseitigung von Fehlern investieren – jedenfalls für den außenstehenden Beobachter.

Nach Durchspielen ihres Modelles mit verschiedenen Parametern ziehen Challet und Le Du (2003) eine Schlussfolgerung, die aus der Perspektive der Qualitätssicherung bei Software bemerkenswert ist:

> From our model we conclude that open source projects converge always faster to a bug-free state for the same set of parameters [...] This finding clearly indicates that closed-source projects must have enough programmers, good enough ones, and have enough users in order to achieve the same software quality in the same amount of time. On the other hand, the quality of open source project programmers does not need to be as high as those of close source projects in order to achieve the same rate of convergence to bug-free programs. (S. 6–7)

Man muss kein gläubiger Anhänger der Open-Source-Community sein, um die grosse Tragweite einer solchen Aussage zu ermessen. Zieht man die Befunde zur Qualifikation von Open-Source-Entwicklern heran, wie sie Studien wie Robles, Scheider, Tretkowski und Weber (2001), FLOSS (2002) und Lakhani, Wolf, Bates und DiBona (2002) geliefert haben, scheint es mehr als naheliegend zu sein, dass Open-Source-Projekte auch in der Praxis die fehlerfreiere Software hervorzubringen vermögen. Unter Berücksichtigung des diskutierten Zusammenhanges zwischen Fehlern von Software und daraus resultierenden Verletzlichkeiten, bedeutet das, dass Software aus Open-Source-Projekten weniger verletzlich ist.[51]

Sicherheit und Software-Ökonomie

Dass die Sicherheit von Software abhängig ist von den ökonomischen Randbedingen ihrer Entstehung und Verbreitung sowie ihres Einsatzes, ist in der Wissenschaft mittlerweile erkannt und akzeptiert.[52]

Ross Anderson, bekannter Fachmann für Sicherheit an der Universität Cambridge in England, stellte unlängst fest, „Open and Closed Systems are Equivalent" (2003). Allerdings schränkt er seine Feststellung ein und lässt sie *so* nur für „an ideal world" gelten. In einem theoretischen Modell zum Wachstum der Systemzuverlässigkeit legt er dar, dass beide Ansätze (Closed Sourceund Open-Source) vergleichbar schnell (oder langsam) an Zuverlässigkeit zunehemende Software hervorbringen

[51] Ausnahmen bestätigen auch hier wieder die Regel. Empirische Befunde jedenfalls stützen diese Aussage. So kann man z.B. anhand der verfügbaren CERT-Informationen zeigen, dass bei einem Vergleich zweier funktional ähnlicher Produkte aus Closed-Source- und aus Open-Source-Produktion, Microsofts Webserver IIS und Open-Source-Webserver Apache, das Open-Source-Produkt qualitativ weniger verletzlich, also sicherer ist (Gehring 2003a). Das gelegentlich gegen einen solchen Vergleich vorgebrachte Argument der geringeren Verbreitung von Open-Source-Produkten greift hier nicht: der IIS hat ca. 23.5%, der Apache ca. 69% Marktanteil (laut http://news.netcraft.com/archives/2004/01/01/january_2004_web_server_survey.html, 8.1. 2004).

[52] Vgl. etwa die Beiträge zu den zwei Workshops on Economics and Information Security (WEIS 2002; WEIS 2003).

würden. In der wirklichen Welt allerdings würde eine Reihe externer Einflussfaktoren insbesondere ökonomischer Art dazu führen, dass die thoretisch erreichbare Äquivalenz des Zuverlässigkeitswachstums in der einen oder anderen Richtung aus der Balance geschoben würde, abhängig vom jeweils betrachteten Parameter. Zur Beurteilung dessen, wie denn in der Praxis sich die Waage zugunsten des einen oder anderen Entwicklungs- und Distributionsmodelles neigen würde, fordert er mehr empirische Untersuchungen,[53] worin ihm sicher Recht zu geben ist. Zwar gibt es eine Reihe empirischer Studien, die zugunsten von Open-Source zu sprechen scheinen. Es dürfte jedoch geboten sein, Vorsicht walten zu lassen, wenn daraus generalisiert werden soll. Zu wenig repräsentativ, zu stark auf einzelne Projekte fokussiert sind die vorliegenden Untersuchungen bisher.

Der Beitrag von Landwehr (2002) hebt hervor, dass aus Sicherheitsgründen ein besserer Informationsfluss notwendig sei, um die Dysfunktionalitäten des Marktes für „information security" zu überwinden. Damit bringt er ein Argument vor, das analog in Gehring (2001; 2003) zu finden ist, was das Problem der asymmetrischen Informationen (s.u.) und seine Auswirkungen betrifft. Auch Landwehr ist der Auffassung, dass quelloffene Software einen Beitrag dazu leisten kann, den Anwendern die rationale Entscheidung *pro* oder *contra* eine bestimmte Software zu erleichtern. Nur so kann man letztlich dem Problem der adversen Auslese (siehe FN 57) entgegenwirken.

In Gehring (2003) stellt der Autor des vorliegenden Beitrages die Frage der Software-Ökonomie insbesondere in den Zusammenhang mit Schutzrechten wie Urheber- und Patentrecht.[54] Da Software sowohl für den Urheberrechtsschutz (auf der textuellen und strukturellen Ebene) als auch für den Patentschutz[55] (auf der funktionalen Ebene) in Frage kommt, müssen die daraus resultierenden ökonomischen Konsequenzen Berücksichtigung finden, wenn es um Sicherheitsfragen geht. Die Tatsache, dass es sich bei Rechten des „geistigen Eigentums" um exklusive Schutzrechte mit sehr weitem Schutzumfang und einer massiven Störwirkung auf den Wettbewerb[56] handelt, garantiert großen Anbietern im Markt potentiell hohe Profite. Dass diese Profite in nicht unerheblichem Umfang realisiert werden, zeigt das Beispiel Microsoft. Auf der anderen Seite kennen weder das Urheber- noch das Patentrecht einen spezifischen Fehlerbegriff und damit verbundene Haftungsregelungen. In der Praxis wirkt sich das so aus, dass Softwarehersteller nur in sehr wenigen Ausnahmefällen für ihre fehlerhaften Produkte in Haftung genommen werden können.

---

53 Instruktiv ist diesbezüglich Wheeler (2003).

54 Erste Betrachtungen dazu wurden in Gehring (2001) vorgestellt. Die ökonomische Struktur des Rechts des „geistigen Eigentums" wird umfassend analysiert in Landes und Posner (2003), die allerdings Software nur am Rande behandeln.

55 Eine umfangreiche Darstellung findet sich bei Stobbs (2000).

56 Schutzrechte für „geistiges Eigentum" blockieren den Imitationswettbewerb, weshalb in der ökonomischen Literatur regelmäßig von Monopolrechten die Rede ist. Vgl. z.B. Landes und Posner (2003, S. 15), die allerdings zurecht darauf hinweisen, dass eine Gleichsetzung von rechtlichem und ökonomischem Monopol in der Regel nicht angebracht ist, wenn der Inhaber des rechtlichen Monopols nicht auch gleichzeitig über große Marktmacht verfügt.

Aus dieser Analyse wird geschlussfolgert, dass Softwarehersteller einerseits große (Profit-) Anreize haben, Software zu produzieren und zu vermarkten, andererseits aber wenig Anreize, (nur) fehlerfreie, sichere Software zu vermarkten. Informationsasymmetrien zwischen Softwareanbieter (bzw. Hersteller) und Softwarenutzer tragen ihren Teil dazu bei, dass ein Markt für sichere Software nicht entstehen kann, der den Bedürfnissen einer global vernetzten IT-Landschaft genügen würde. „Adverse selection“[57] sorgt dafür, dass sich sichere Produkte nicht in der Breite etablieren können, die für ein signifikant höheres Sicherheitsniveau notwendig wäre. Intellectual Property Rights verschärfen das Problem insofern, als dass Selbsthilfe, z.B. in Form eines „reverse compilation“[58] zu Evaluationszwecken, rechtlich weitgehend für unzulässig erklärt worden ist.

Im Anschluss an diese Feststellung fragt Gehring (2003), inwiefern Open-Source-Software geeignet sein könnte, Abhilfe zu schaffen. Zwar löst Open-Source-Software nicht das Problem der unzureichenden Haftung bei Software, aber sie beseitigt in erheblichem Maße die Informationsasymmetrien zwischen Hersteller und Anwender:

- Anwender haben Einblick in den Quellcode, erfahren also, was genau sie erwerben;
- Sicherheitsmängel können im Quellcode evaluiert werden, ggf. als Dienstleistung durch Dritte.

Der Quellcode bietet damit eine Signalfunktion,[59] wie sie zur Überwindung von Informationsasymmtrien dringend benötigt wird: Wenn ein Anwender sich vor einem geplanten Einsatz überzeugen will, ob ein proprietäres Produkt oder ein Open-Source-Produkt vergleichbarer Funktionalität seinen Zwecken besser genügt, ist er zu einer Evaluation der Produkteigenschaften gezwungen. Zur Evaluation stehen für den Anwender unterschiedliche Signale zur Auswertung bereit: (1) Werbeinfor-

[57] Das Problem der adversen Auslese (adverse selection) entsteht in Märkten, in denen die potentiellen Vertragspartner vor Vertragsschluss über asymmetrische Informationen verfügen: Einer der beiden potentiellen Vertragspartner, der Anbieter eines Produkts oder einer Leistung, weiss typischerweise mehr über die Qualität seines Angebots als der potentielle Käufer, dem die Beziehung zwischen Qualität und Preis vor dem Erwerb und Ausprobieren des Angebots nicht objektiv klar ist. Konfrontiert mit anscheinend vergleichbaren Angeboten und unterschiedlichen Preisen wird er maximale Leistung bei minimalen Kosten anstreben und sich für das preiswerteste Angebot entscheiden. Wenn Anbieter höherer Qualitäten nicht in der Lage sind, auf Grund ihrer Selbstkosten, vergleichbare Preise anzubieten, werden sie nicht wettbewerbsfähig sein. Eine Lösung besteht für dieses Dilemma dann darin, die Qualität des Angebots zu reduzieren, um ebenfalls niedrige Preise anbieten zu können. Auf diese Weise sinkt die Menge der im Markt angebotenen Qualitätsprodukte und nur die „schlechten Qualitäten“ werden gehandelt. Eine Lösungsmöglichkeit ist z.T. das *signalling*, wie im Text diskutiert.

[58] Unter „reverse compilation“ versteht man die Rekonstruktion der programmsprachlichen Präsentation eines binären Programmes in einer für den Menschen lesbaren Form. „Reverse compilation“ ist nicht zu verwechseln mit „reverse engineering“, bei dem es im Kern um die Sichtbarmachung der zugrundeliegenden Ideen und Konzepte einers Programmes geht. Allerdings setzt „reverse engineering“ in der Regel eine „reverse compilation“ voraus, sodass die Grenzen fließend sind.

[59] Mithilfe solcher Funktionen werden dem potentiellen Kunden Qualitätsmerkmale, bzw. allgemein Produkteigenschaften signalisiert, die nicht ohne weiteres beobachtet werden können. So signalisiert etwa eine freiwillige Garantie das Vertrauen eines Anbieters in die Haltbarkeit seines Produkts und eine Marke überträgt Erfahrungen aus der Vergangenheit auf Erwartungen an die Zukunft; Patente können zur Signalisierung der Innovativität eines Anbieters dienen, usw.

mationen des Anbieters; (2) Befragung von Experten (oft indirekt, via Artikel in der Fachpresse); (3) Ausprobieren des Produkts; (4) Begutachtung der „inneren Werte", d.h. Aufbau, Ausführung, Dokumentation usw. des Programmcodes.

Vergleicht man hinsichtlich dieser Optionen proprietäre (Closed-Source-) Software und Open-Source-Software, so kann man zu folgender Gegenüberstellung[60] gelangen:

Informationsbeschaffung	Closed-Source	Open-Source
aus Werbung	ja	ja (zum Teil)
Expertenmeinungen	ja	ja
Erfahrung durch Ausprobieren	nein	ja
eigene Auditierung	nein	ja
unabhängige Auditierung	selten	ja

Informationsbeschaffung vor dem Erwerb/Einsatz

Die Informationsasymmetrien zwischen Anbieter und Anwender (Kunde) können mithin auf Grund der Lizenzbedingungen rechtlich und, dank des einsehbaren Quellcodes bei Open-Source-Software, auch faktisch weitestgehend abgebaut werden. Gerade der Faktor Sicherheit lässt sich nur sehr schlecht[61] bewerten, wenn der Quellcode nicht zur Verfügung steht.[62] Gerade hier kommt der Signalfunktion (4), d.h. den offenliegenden Quellcodes bei Open-Source-Software, immense Bedeutung zu.

Erst mit der Beseitigung der Informationsasymmetrien kann ein echter Qualitätswettbewerb[63] entstehen, der zu dem gewünschten Ergebnis führt: qualitativ bes-

60 Ausnahmen für Großkunden, Regierungen usw. sind zwar bei proprietärer Closed-Source-Software im Einzelfall hilfreich, im Normalfall jedoch unerreichbar.

61 Im Prinzip kann bei nicht vorliegendem Quellcode nur eine quantitative Bewertung durch massive Penetrationstests („Zerstörungsprüfung") erfolgen, die statistische Aussagen über die Produktsicherheit zulässt.

62 Aus diesem Grunde ist bei einer Sicherheitszertifizierung nach *Common Criteria (CC)* oder *Information Technology Security Evaluation Criteria (ITSEC)* in den höheren Evaluationsstufen die Vorlage des Quellcodes unverzichtbar.

63 Der Wettbewerb wird zwar im Markt für Produkte, aber nicht zwischen klassischen Firmen, sondern zwischen Open-Source-Projekten und anderen Open-Source-Projekten bzw. Firmen ausgetragen. Aus der Perspektive eines Marktes in dem dominierende Anbieter von wettbewerbswidrigem Verhalten nicht effektiv abgehalten werden können (vgl. den Ausgang des Microsoft-Prozesses und Microsofts anschließendes Verhalten), ist dieser Umstand von großer Bedeutung. Im Unterschied zu einer Firma kann ein Projekt (das unterschiedlichste Institutionen einschließt, darunter auch Firmen) nicht ohne weiteres aus dem Markt verdrängt werden, etwa durch Aufkauf oder kostspielige Gerichtsverfahren. Potentielle Wettbewerber im Markt für Produkte greifen stattdessen die rechtliche Basis der Projekte an (siehe den Prozess SCO vs. IBM in dem die Copyright-Konformität der GNU Public License in Frage gestellt wird), setzten Patente als „strategische Waffen" ein (etwa in Standardisierungsprozessen), installieren proprietäre Schnittstellenformate die eine Interoperabilität von Open-Source-Produkten mit proprietären Produkten erschweren oder versuchen, die offene PC-Plattform gegen Modifikationen von Seiten des Computerbesitzers abzusichern (sog. „trusted platforms"), was sich negativ auf Open Source und andere Wettbewerber auswirken könnte

sere (sicherere) Produkte setzen sich durch. Ausgehend von dieser Einsicht schlägt Gehring (2003) zur Erhöhung der IT-Sicherheit ein Risikomanagement vor, dessen Fundament Open-Source-Software sein sollte. Darauf sollte ein umfassendes System des Source-Code-Auditing aufgebaut werden, das z.B. bei Beschaffungsentscheidungen der Verwaltung eine marktstimulierende Rolle übernehmen könnte. Insofern Bestimmungen zum Schutz des „geistigen Eigentums" (etwa im Bereich des „reverse engineering" usw.) den Sicherheitsbedürfnissen entgegenstehen, sollten sie geeignet modifiziert werden.

Franke und von Hippel (2003) haben einen anderen Problembereich des klassischen Softwaremarktes untersucht: Nutzerbedürfnisse sind, in Abhängigkeit von der Art der Produkte, in unterschiedlichem Umfang heterogen. Ein Anbieter muss diese Heterogenität erforschen und in seiner Produktstrategie berücksichtigen. Dazu werden beispielsweise Märkte segementiert und je Segment die angebotenen Produkte in ihren Eigenschaften und Preisen differenziert.[64] Sollte es aus Anbietersicht zu kostspielig sein, bestimmte Marktsegmente zu bedienen, werden dafür keine passenden Produkte angeboten (Franke und von Hippel 2003, S. 1199). Die potentiellen Kunden können ihre Nachfrage dann nicht über den Markt decken.

Mit dem Open-Source-Modell ist es nun möglich, einen Teil der Entwicklungskosten für einen Markt mit sehr heterogenen Bedürfnissen auf die Anwender zu übertragen, wodurch erheblich mehr Marksegmente bedient werden können. Das trifft auch auf die Sicherheitseigenschaften von Softwareprodukten zu (S. 1200). Diese Überlegungen wurden von Franke und von Hippel (2003) am Beispiel des Apache-Webservers empirisch überprüft. Die Befunde ergaben,

> [...] that user needs for Apache security functionality are in fact highly heterogeneous. Next, we find that many Apache users are not fully satisfied by existing standard Apache security offerings. When we apply a very conservative measure of willingness to pay (WTP) (an 80% deflation of expressed willingness to pay), we find that the average Apache user is willing to pay a considerable amount (over US$ 5000 per user and over US$ 160 million in aggregate) to have their individual needs for the security functions of Apache met to their total satisfaction. (S. 1200)

Die Tatsache, dass der Source Code des Apache-Webservers als Open-Source den Anwendern zur Verfügung steht, ermöglicht es ihnen, je nach Investitionsbereitschaft, in die Verbesserung der Sicherheit zu investieren. Ein Teil der Anwender setzt die Möglichkeit in die Tat um und modifiziert den Apache-Code mit dem Ergebnis, dass ihre Sicherheitsanforderungen besser befriedigt werden können:

(Anderson 2003; Koenig und Neumann 2003). Doch besteht da keine Zwangsläufigkeit; vielmehr bieten „trusted platforms" gerade auch in Verbindung mit dem Open-Source-Modell ein großes Potential, IT-Systeme sicherer zu machen, ohne den Wettbewerb auszuschließen (Kuhlmann und Gehring 2003, S. 200–202).

64 Das einfachste Beispiel ergibt sich aus den unterschiedlichen Zahlungspräferenzen potentieller Autokäufer. Für Kunden mit niederiger Kaufbereitkraft werden einfachere Produkte (Automodelle) angeboten, für Kunden mit hoher Kaufkraft hochwertige Produkte (Luxusautos). In der Grundfunktionalität unterscheiden sich die Produkte jedoch nicht wesentlich: „Auto ist Auto!" (und alle fahren sie auf Rädern). Mit der Qualitäts-/Preisdifferenzierung kann der Anbieter einen wesentlich größeren Kundenkreis bedienen und dadurch höhere Gesamtprofite erwirtschaften.

> When we compare responses from innovating and non-innovating users, we find that users that modify the standard product report significantly higher satisfaction levels than those that do not. (S. 1200)

Hinzu kommt, dass auch Anwender mit geringerer (oder ohne) Investitionsbereitschaft von den Modifikationen profitieren, wie es bei der Open-Source-Methode durch die Weitergabe des verbesserten Quellcodes die Regel ist.[65] Unter Berücksichtigung der Heterogenität anderer Märkte, die der im Apache-Fall vergleichbar ist, schließen Franke und von Hippel, dass der Einsatz von „toolkits for user innovation," wie beispielsweise die Zurverfügungstellung von Software im Quellcode und entsprechende Lizenzierung, dazu führen könne, die Nutzerbedürfnisse – auch ihre Sicherheitsbedürfnisse – besser zu befriedigen als durch klassische Produktentwicklung und -vermarktung (2003, S. 1200).

Zusammenfassung und Schlussfolgerung

Im vorliegenden Aufsatz wurde versucht, die vier wesentlichen Pfeiler der bisherigen Debatte um die Sicherheit von Open-Source und ihrer Hintergründe zu identifizieren, sowie auf ein erhebliches Defizit im Bereich der Psychologie und Soziologie der Softwareentwicklung hinzuweisen.

Dabei wurde deutlich gemacht, dass die Debatte zum Teil zwar auch auf polemischen Elementen aufsetzt, zum überwiegenden Teil jedoch (aus wissenschaftlicher Sicht) gerechtfertigt erscheint.

Zwei Pfeiler der Debatte, nämlich:

- der regelmäßig behauptete Zusammenhang zwischen dem quelloffenen Prozess der Softwareerstellung/-distribution von Software aus technischer Sicht sowie

[65] Dieser Befund stützt eine Vermutung des Autors, dass „free riding", also Trittbrettfahrerei, in der ökonomischen Literatur immer als problematisch angesehen (vgl. Fritsch, Wein und Ewers 2001, S. 90 ff.; Macho-Stadler und Pérez-Castrillo 2001, S. 90; Varian 2002), in Netzwerkumgebungen in Maßen durchaus wünschenswert sein kann, wo es um die Sicherheit des Netzwerkes geht (zumal wenn, wie im Apache-Beispiel, ganz offensichtlich individuelle Investitionsanreize vorhanden sind, obwohl das zu erwartende „free riding" ja vorher bekannt ist – aus den Apache-Lizenzbedingungen). Aus demselben Grund ist es sinnvoll, zur Bekämpfung einer Epidemie Medikamente kostenlos abzugeben, wenn Externalitäten existieren (akute Ansteckungsgefahr). Ein weiteres ökonomisches Argument hat Dirk Kuhlmann vorgebracht: Angenommen, ein Anwender eines bestimmten Open-Source-Programmes stellt einen Fehler fest, sucht und findet den Fehler im Quellcode und beseitigt ihn. Handelt es sich um einen sicherheitsrelevanten Fehler, dessen Auftreten auf anderen Systemen ihm selbst einen Vorteil verschaffen kann, steht er vor einem Dilemma: Soll er den Fehler und dessen Beseitigung („patch") an die Entwickler weitermelden? Tut er das, verliert er auf der einen Seite den erwähnten Vorteil; auf der anderen Seite wird er, vorausgesetzt, die Entwickler beseitigen den Fehler ein für alle Mal, in Zukunft bei neuen Programmversionen selbst keinen Beseitigungsaufwand (d.h. Kosten) mehr haben. Hinzu kommt, dass andere Anwender den Fehler ihrerseits finden und melden könnten, d.h. sein Vorteil ohnehin nur kurz währen würde. „Unter der Annahme, dass die meisten Benutzer ehrliche Leute sind und es für sie billiger ist, den Bug zu melden und damit das Problem ein für alle Mal aus der Welt zu schaffen, gibt es also einen ökonomischen Anreiz zur Kooperation im IT-Sicherheitsbereich. Das ist eine gute Sache, denn fehlende Kooperation ist eines der Hauptprobleme in dieser Sparte." (persönliche Kommunikation mit D. Kuhlmann 2004)

- die ökonomischen Randbedingungen einer klassischen, marktwirtschaftlichen Organisation der Softwareherstellung/-distribution vs. der Open-Source-Methode

und ihre Einflüsse auf Qualität/Sicherheit der resultierenden Softwareprodukte wurden anhand der aktuellen wissenschaftlichen Literatur diskutiert. Sowohl die vorgestellten Befunde aus empirischen Untersuchungen als auch die theoretischen Überlegungen lassen es in ihrer Gesamtheit als gerechtfertigt erscheinen, die Frage danach, ob denn Open-Source bessere – und sicherere – Software hervorbringen würde, mit *„ja, oft"* zu beantworten.

Eine solche Erkenntnis müsste in Anbetracht der durch fehlerhafte Software verursachten volkswirtschaftlichen Schäden auf politischer Ebene normalerweise zu Schlussfolgerungen führen:

Statt das Closed-Source-Modell bevorzugt zu behandeln, z. B. durch urheber- und patentrechtliche Unterstützung,[66] *müsste die breite Anwendung eines auf dem Open-Source-Modell basierenden Risikomanagements durch die Schaffung geeigneter Anreizstrukturen gefördert werden.*[67]

Jedenfalls müssten gewichtige und prüfbare Gegenargumente ins Feld geführt werden, um Forderungen nach einem solchen Paradigmenwechsel in Anbetracht der mittlerweile vorliegenden wissenschaftlichen Befunde zu deligitimieren. Die Beweislast liegt inzwischen, scheint es, auf Seiten der Befürworter des Closed-Source-Modells.

Literatur

Anderson, Ross J. (2001): *Security Engineering: A Guide to Building Dependable Distributed Systems,* New York: John Wiley & Sons

Anderson, Ross J. (2001a): *Why Information Security is Hard – An Economic Perspective*, in: Proceedings of the Seventeenth Computer Security Applications Conference 2001, Los Alamitos, CA: IEEE Comput. Society, online http://www.cl.cam.ac.uk/ftp/users/rja14/econ.pdf

Anderson, Ross J. (2003): *Open and Closed Systems are Equivalent (that is, in an ideal world)*, online http://www.cl.cam.ac.uk/ftp/users/rja14/toulousebook.pdf

Beck, Kent (2000): *Extreme Programming Explained*, Boston, MA: Addison-Wesley

Bollier, David (2003): *Silent Theft: The Private Plunder of Our Common Wealth*, New York und London: Rouledge

Boyle, James (1996): *Shamans, Software* & *Spleens: Law and the Construction of the Information Society*, Cambridge, MA & London: Harvard University Press

Boehm, Barry W. (1981): *Software Engineering Economics*, Englewood Cliffs, NJ: Prentice Hall

[66] Vgl. Lutterbeck, Horns und Gehring (2000) und Gehring (2001; 2003a).

[67] Dass dabei durchaus im Einzelfall zu differenzieren ist, wird aus den Einsichten von McCormack (2001) und Payne (2002) klar. Und welche Anreizstrukturen erfolgversprechend sein werden, lässt sich nicht mit Gewissheit vorhersagen. Die Bereitstellung von Infrastrukur durch die DARPA im Auftrage des US-Verteidigungsministeriums zum Zwecke „at drawing skilled eyeballs to the thankless task of open-source security auditing" (Poulsen 2004) ist jedenfalls gescheitert.

Branscomb, Anne Wells (1994): *Who Owns Information? From Privacy to Public Access*, New York: Basic Books

Campbell-Kelly, Martin und William Aspray (1996): *Computer: A History of the Information Machine*, New York: Basic Books

Carini, Brian und Benoit Morel (2002): *Dynamics and Equilibria of Information Security Investment*, in: WEIS (2002)

Clapes, Anthony L. (1993): *Softwars: The Legal Battles for Control of the Global Software Industry*, Westport, CT: Quorum Books

Cockburn, Alister (2002): *Agile Software Development*, Boston, MA: Addison-Wesley

Correa, Carlos M. (2000): *Intellectual Property Rights, the WTO and Developing Countries. The TRIPS Agreement and Policy Options*, London und New York: Zed Books

Dignatz, Eitel (1999): *Sicherheit durch Open Source*, Gastkommentar in: Computerwoche vom 13. 08. 1999,
online http://www.dignatz.de/d/spotlight/artikel/oss+sicherheit_19990813_002.html (7.1.2004)

Dorchester, Dave (1999): *Why License Software Engineers?* In: IEEE Software, Vol. 16, No. 2 (March/April 1999), S. 101–102

Drahos, Peter und John Braithwaite (2002): *Information Feudalism: Who Owns the Knowledge Economy?* New York: The New Press

Economist (2003): *Microsoft at the power point*, in: Economist News, 11. September 2003,
online http://www.economist.com/business/displayStory.cfm?story_id=2054746

Fisk, Mike (2002): *Causes & Remedies for Social Acceptance of Network Insecurity*, in: WEIS (2002)

FLOSS (2002): *Free/Libre and Open Source Software: Survey and Study*, Forschungsbericht, International Institute of Infonomics, University of Maastricht, The Netherlands und Berlecon Research GmbH, Berlin, Deutschland, June 2002, http://www.infonomics.nl/FLOSS/report/

Forno, Richard (2003): *Overcoming ‚Security By Good Intentions'*, in: The Register News vom 9. Juni 2003,
online http://www.theregister.co.uk/content/55/31094.html

Franke, Nikolaus und Eric von Hippel (2003): *Satisfying heterogeneous user needs via innovation toolkits: the case of Apache security software*, in: Research Policy, Jg. 32, Nr. 7, S. 1199–1215,
online http://dx.doi.org/10.1016/S0048-7333(03)00049-0; Vorabversion erschienen als MIT Sloan School of Management Working Paper # 4341-02, (January 2002)
online http://web.mit.edu/evhippel/www/ApacheHeteroWP.pdf

Frailey, Dennis J. (1999): *Licensing Software Engineers*, in: Communications of the ACM, Vol. 42, No. 12, S. 29–30

Fritsch, Michael, Thomas Wein und Hans-Jürgen Ewers (2001): *Marktversagen und Wirtschaftspolitik*, 4. Aufl., München: Vahlen

Gehring, Robert A. (2001): *Software Patents – IT-Security at Stake?* Präsentation auf dem Kongress „Innovations for an e-Society. Challenges for Technology As-

sessment“, Berlin, 17.–19. Oktober 2001,
online http://ig.cs.tu-berlin.de/ap/rg/2001-10/index.html.

Gehring, Robert A. (2003): *2003 (1) The Journal of Information, Law and Technology (JILT)*,
online http://elj.warwick.ac.uk/jilt/03-1/gehring.htm

Gehring, Robert A. (2003a): *Open Source Software – Sicherheit im Spannungsfeld von Ökonomie und Politik*, Beitrag auf dem CAST-Workshop „Sicherheit mit Open Source?“, Darmstadt, 20. März 2003, Präsentation und Text via http://ig.cs.tu-berlin.de/ap/rg/

Gehring, Robert A. (2003b): *Business Case: Open Source Security*, Präsentation auf der Euroforum-Jahrestagung „Sicherheit 2003“, Hamburg, 11.–14. November 2003,
online http://ig.cs.tu-berlin.de/ap/rg/2003-11/Gehring-Euroforum-OSSSecurity-112003.pdf.

Goltzsch, Patrick (2003): *Linux beschleunigt seinen Vormarsch*, in: FTD News vom 29. Dezember 2003,
online http://www.ftd.de/tm/it/1072525171182.html?nv=hpm.

Gordon, Kawrence A. und Martin P. Loeb (2002): *The Economics of Information Security Investment*, in: ACM Transactions on Information and System Security, Vol. 5, No. 4 (November 2002), S. 438–457.

Greenwals, Michael, Carl A. Gunter, Björn Knutsson, Andre Scedrov, Jonathan M. Smith und Steve Zdancevic (2003): *Computer Security is Not a Science (but it should be)*,
online http://www.cis.upenn.edu/~stevez/papers/GGKS+03.pdf

heise (2004): *Bill Gates: Microsofts Daten-Armbanduhr auch in Europa*, in: heise newsticker vom 9. Januar 2004,
online http://www.heise.de/newsticker/data/jk-09.01.04-001/

Hillmann, Karl-Heinz (1994): *Wörterbuch der Soziologie*, 4. Aufl., Stuttgart: Alfred Kröner

Hofmann, Jan (2002): *Free software, big business?* Forschungsbericht Nr. 32, Deutsche Bank Research, Frankfurt am Main,
online http://www.dbresearch.de/PROD/PROD0000000000047532.pdf

Hoglund, Greg (2002): Security Band-Aids: *More Cost-Effective than „Secure“ Coding*, in: IEEE Software, Vol. 19, No. 6 (November/December 2002), S. 57–58

Hohmann, Luke (1996): *Journey of the Software Professional: A Sociology of Software Development*, Upper Saddle River, NJ: Prentice Hall PTR

Imparato, Nicholas, Hrsg. (1999): *Capital for Our Time: The Economic, Legal, and Management Challenges of Intellectual Capital*, Stanford, CA: Hoover Institution Press

Ishii, Kei und Bernd Lutterbeck (2002): *Der Microsoft-Prozess*, in: Alexander Roesler und Bernd Stiegler (Hrsg.): *Microsoft: Medien. Macht. Monopol*, S. 130–153, Frankfurt am Main: Suhrkamp,
online http://ig.cs.tu-berlin.de/bl/072/IshiiLutterbeck-MicrosoftProzess-2002.pdf

Jonietz, Erika (2004): *Valid Voring?* In: MIT Technology Review, Jg. 107, 2004, Nr. 1 (Februar), S. 74–75

Jonsson, Erland, Lars Strömberg und Stefan Lindskog (2000): *On the Functional Relation Between Security and Dependability Impairments*, in: Proceedings of the 1999 New Security Paradigm Workshop, September 1999, Ontario, S. 104–111

Kalam, A. P. J. Abdul (2003): *Convergence of Technologies*, Rede für das International Institute of Information Technology, 28. Mai 2003,
online http://presidentofindia.nic.in/S/html/speeches/others/may28_2003_2.htm

Kamerling, Erik (2003): *Three Questions for the October 8, 2003 Top 20 Briefing*,
online http://www.sans.org/top20/overview03.pdf.

Knight, John C. (2002): *Dependability of Embedded Systems*, in: Proceedings of ICSE'02, 19.–25. May 2002, Orlando, Florida, S. 685–686

Koenig, Christian und Andreas Neumann (2003): *Standardisierung und EG-Wettbewerbsrecht – ist bei vertrauenswürdigen Systemumgebungen wettbewerbspolitisches Misstrauen angebracht?* In: WuW 11/2003, S. 1138–1152

Köhntopp, Kristian, Marit Köhntopp und Andreas Pfitzmann (2000): *Sicherheit durch Open Source? Chancen und Grenzen*, in: Datenschutz und Datensicherheit (DuD), Jg. 24, Nr. 9, S. 508–513

Kolawa, Adam (2002): *Certification Will Do More Harm Than Good*, in: IEEE Computer, Vol. 35, No. 6, S. 34–35

Krsul, Ivan, Eugene Spafford und Mahesh Tripunitara (1998): *Computer Vulnerability Analysis*, 6. Mai 1989. Technical Report COAST TR98-07, COAST Laboratory, Purdue University, West Lafayette, IN,
online http://ftp.cerias.purdue.edu/pub/papers/ivan-krsul/krsul9807.pdf

Kuhlmann, Dirk und Robert A. Gehring (2003): *TCPA, DRM, and Beyond*, in: Eberhard Becker, Willms Buhse, Dirk Günnewig und Niels Rump (Hrsg.): Digital Rights Management: Technological, Economic, Legal and Political Aspects, Berlin, Heidelberg, S. 178–205, New York: Springer Verlag

Laffont, Jean-Jacques und David Martimort (2002): *The Theory of Incentives*, Princeton, NJ und Oxford: Princeton University Press

Lakhani, Karim R., Bob Wolf, Jeff Bates und Chris DiBona (2002): *The Boston Consulting Group Hacker Survey*,
online http://www.osdn.com/bcg/bcg-0.73/BCGHackerSurveyv0-73.html

Landes, William M. und Richard A. Posner (2003): *The Economic Structure of Intellectual Property Law*, Cambridge, MA und London: Belknap/Harvard University Press

Landwehr, Carl (2002): *Improving Information Flow in the Information Security Market*, in: WEIS (2002)

Lawton, George (2002): *Open Source Security: Opportunity or Oxymoron?* In: IEEE Computer, Vol. 35, No. 3, S. 18–21

Lemley, Mark A., Peter S. Menell, Robert P. Merges und Pamela Samuelson (2002): *Software and Internet Law*, 2. Aufl., New York: Aspen Publishers

Lessig, Lawrence (1999): *Code and Other Laws of Cyberspace*, New York: Basic Books

Levy, Leon S. (1987): *Taming the Tiger – Software Engineering and Software Economics*, New York: Springer Verlag

Levy, Steven (2001): *Hackers: Heroes of the Computer Revolution*, New York: Penguin Books

Li, Wei (2002): *Security Model of Open Source Software*, online http://www.cs.helsinki.fi/u/campa/teaching/oss/papers/wei.pdf.

Litman, Jessica (2001): *Digital Copyright*, Amherst, N.Y.: Prometheus Books

Luqi und Joseph A. Gogue (1997): *Formal Methods: Promises and Problems*, in: IEEE Software, Vol. 14, Iss. 1, Januar 1997, S. 73–85

Lutterbeck, Bernd, Axel H. Horns und Robert A. Gehring (2000): *Sicherheit in der Informationstechnologie und Patentschutz für Softwareprodukte – Ein Widerspruch?,* Kurzgutachten erstellt im Auftrag des Bundesministeriums für Wirtschaft und Technologie, Berlin, online http://www.sicherheit-im-internet.de/download/Kurzgutachten-Software-patente.pdf

Macho-Stadler, Inés und J. David Pérez-Castrillo (2001): *An Introduction to the Economics of Information: Incentives and Contracts*, Oxford und New York: Oxford University Press

Macrina, Francis L. (1995): *Scientific Integrity: An Introductory Text with Cases*, Washington, D.C.: ASM Press

Maggs, Peter B., John T. Soma und James A. Sprowl (2001): *Internet and Computer Law: Cases-Comments-Questions*, St. Paul, MN: West Group

Meadows, Catherine und John McLean (1999): *Security and Dependability: Then and Now*, in: Proceedings of Computer Security, Dependability, and Assurance: From Needs to Solutions, 7–9 July 1998 and 11–13 November 1998, York, UK & Williamsburg, VA, USA, S. 166–170. Los Alamitos, CA: IEEE Comput. Society

McChesney, Robert W., Ellen Meiksins Wood und John Bellamy Foster (1998): *Capitalism and the Information Age: The Political Economy of the Global Communication Revolution*, New York: Monthly Review Press

Mundie, Craig (2003): *Security: Source Access and the Software Ecosystem*, online http://www.microsoft.com/resources/sharedsource/Security/SourceAccess.mspx

Mustonen, Mikko (2003): *Copyleft – the economics of Linux and other open source software*, in: Information Economics and Policy, Vol. 15, S. 99–121

Myers, Glenford J. (1976): *Software Reliability: Principles and Practice*, New York: John Wiley & Sons

Myers, Glenford J. (1979): *The Art of Software Testing*, New York: John Wiley & Sons

National Research Council (2000): T*he Digital Dilemma. Intellectual Property in the Information Age*, Washington, D.C.: National Academy Press

National Research Council (1991): *Intellectual Property Issues in Software*, Washington, D.C.: National Academy Press

Office of Technology Assessment (1992): *Finding a Balance: Computer Software, Intellectual Property and the Challenge of Technological Change*, OTA-TCT-527, Washington, D.C.: Government Printing Office

Oppliger, Rolf (2003): *Sicherheit von Open Source Software*, in: Datenschutz und Datensicherheit (DuD), Jg. 27, 2003, Nr. 11, S. 669–675, http://www.dud.de/dud/documents/dud11-03.pdf

Parnas, David L., A. John van Schouwen und Shu Po Kwan (1990): *Evaluation of Safety-Critical Software*, Communications of the ACM, Vol. 33, Nr. 6, S. 636–648

Poulsen, Kevin (2004): *DARPA-funded Linux security hub withers*, in: Securityfocus News vom 30. Januar 2004,
online http://www.securityfocus.com/news/7947

Raymond, Eric S. (1999): *The Cathedral and the Bazaar*, S. 27–78, in: Eric S.Raymond: The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary, Sebastopol, CA: O'Reilly,
online http://www.openresources.com/documents/cathedral-bazaar/main.html

Robles, Gregorio, Hendrik Scheider, Ingo Tretkowski und Niels Weber (2001): *Who Is Doing It? A Research on Libre Software Developers*, Forschungsbericht, August 2001, Forschungsgebiet Informatik und Gesellschaft, Technische Universität Berlin,
online http://ig.cs.tu-berlin.de/s2001/ir2/ergebnisse/OSE-study.pdf

Ryan, Michael P. (1998): *Knowledge Diplomacy: Global Competition and the Politics of Intellectual Property*, Washington, D.C.: Brookings Institution Press

Sandhu, Ravi (2003): *Good-Enough Security*, in: IEEE Internet Computing, Vol. 7, No. 1 (January/February 2003), S. 66–68

Scacchi, Walt (2002): *Is Open Source Software Development Faster, Better, and Cheaper than Software Engineering?*, Paper presented at the 2nd ICSE Workshop on Open Source Software Engineering, May 2002, Orlando, FL

Schneier, Bruce (1996): *Angewandte Kryptographie: Protokolle, Algorithmen und Sourcecode in C*, 2. Aufl., Reading, MA: Addison Wesley

Schneier, Bruce (2000): *Secrets and Lies. Digital Security in a Networked World*, New York: John Wiley & Sons

Schneier, Bruce (2002): *Computer Security: It's the Economics, Stupid*, in: WEIS (2002)

Schulzki-Haddouti, Christiane (2001): *Das Ende der Schweigsamkeit: EU-Parlament verabschiedet Echelon-Untersuchungsbericht*, in: c't 19/2001, S. 44

Myung, Seung eun (2003): *Korea jettisons Windows for Linux*, in: silicom.com News, 1. Oktober 2003,
online http://www.silicon.com/news/500011-500001/1/6225.html?rolling=1

Stamelos, Ioannis, Lefteris Angelis, Apostolos Oikonomou und Georgios L. Bleris (2002): *Code quality analysis in open source software development*, in: Info Systems Journal, Vol. 12, Iss. 1, S. 43–60,
online http://www.blackwell-synergy.com/links/doi/10.1046/j.1365-2575.2002.00117.x/abs/

Stark, Jacqueline (2002): *Peer Reviews as a Quality Management Technique in Open-Source Software Development Projects*, in: Proceedings of Software Quality – ECSQ

2002: 7th International Conference, Helsinki, Finland, June 9–13, 2002, S. 340–350

Stobbs, Gregory A. (2000): *Software Patents*, 2. Aufl., Gaithersburg und New York: Aspen Law & Business

Tan, Bernard C. Y., H. Jeff Smith, Mark Keil und Ramiro Montalegre (2003): *Reporting Bad News About Software Projects: Impact of Organizational Climate and Information Asymmetry in an Individualistic and a Collectivistic Culture*, in: IEEE Transactions on Engineering Management, Vol. 50, No. 1, February 2003, S. 64–77

Thierer, Adam und Clyde Wayne Crews, Jr. (2003): *What's Yours is Mine: Open Access and the Rise of Infrastructure Socialism*, Washington, D.C.: Cato Institute

Tockey, Steve (1997): *A Missing Link in Software Engineering*, in: IEEE Software, Vol. 14, No. 6 (November/December 1997), S. 31–36

Tripp, Leonard L. (2002): *Benefits of Certification*, in: IEEE Computer, Vol. 35, No. 6, S. 31–33

Varian, Hal (2002): *System Reliability and Free Riding*, in: WEIS (2002).

Weinberg, Gerald M. (1971): *The Psychology of Computer Programming*, New York: Van Nostrand Reinhold

Wheeler, David A. (2003): *Why Open Soure Software / Free Software (OSS/FS)? Look at the Numbers!*, Abschnitt 6: Security,
online http://www.dwheeler.com/oss_fs_why.html

Wei-Ming, Tu (1993): *Way, Learning, and Politics: Essays on the Confucian Intellectual*, Albany, NY: State University of New York Press

WEIS (2002): *Proceedings of the 1st Workshop on Economics and Information Security*,
online http://www.sims.berkeley.edu/resources/affiliates/workshops/econsecurity/

WEIS (2003): *Proceedings of the 2nd Workshop on Economics and Information Security*,
online http://www.cpppe.umd.edu/rhsmith3/index.html

Zao, Luyin und Sebastian Elbaum (2003): *Quality Assurance und the open source development model*, in: The Journal of Systems and Software, Vol. 66, No. 1, S. 65–75

Zetter, Kim (2004): *Open-Source E-Voting Heads West*, in: Wired News, 21. Januar 2004,
online: http://www.wired.com/news/evote/0,2645,61968,00.html

Open Source und offene Standards

DIRK KUHLMANN

In einem 1996 erschienenen Artikel mit dem Titel „The Rise and Fall and Rise of Standardization“ (Cargill 1996) diskutiert Carl Cargill die Frage, ob Standardisierung im Bereich der Datenverarbeitung einer Art von historischen Zyklen unterworfen ist. Ausgangspunkt seiner Überlegungen ist, dass sich die Geschichte der Standardisierung in diesem Bereich auf verschiedenen Abstraktionsniveaus zu wiederholen scheint. Der Kampf um Hardwarekompatibilität der 60er und 70er Jahre wurde, so der Autor, durch Konflikte um interoperable Software und Daten in den 80er und 90er Jahren abgelöst. Dieser mache soeben einer neuen Runde von Auseinandersetzungen Platz, die nun auf der Ebene der verteilten Services und der hierzu notwendigen Protokolle stattfänden. Der Eindruck der Zyklizität, der Wiederkehr des immer Gleichen, rührt Cargill zufolge vor allem daher, dass ein bestimmtes Muster wiederkehrt, nämlich die Entwicklung fort von geschlossen und hin zu offenen Lösungen.

Sein Erklärungsvorschlag für die (IT-Systemen vermeintlich innewohnende) Tendenz hin zu mehr Offenheit lautet, dass der Markt keineswegs so dumm sei wie oft angenommen. Offenheit sei nämlich nicht allein als technisches Kriterium zu verstehen. Ihr Grad sei auch und vor allem an der Anzahl der Individuen zu bemessen, denen ein nutzbringender Umgang mit Technologie eröffnet wird, und der Markt tendiere dahin, diese Anzahl zu erhöhen. Auf welche Art auch immer sich Standards im IT-Bereich jeweils durchgesetzt hätten, die Richtung hin zu einer in diesem Sinne verstandenen Offenheit sei das allen gemeinsame Leitmotiv. Standardisierung an sich finde dagegen zu allen Zeiten statt. Nicht sie selbst, sondern ihre Popularität und Publizität sei es, die konjunkturell schwanke.

Cargills Darstellung dient im Folgenden als Kontrastfolie für einen genaueren Blick auf das Verhältnis zwischen offener Software und offenen Standards. Sein Versuch einer historischen Zusammenschau wird dabei um eine zeitgeschichtliche Schleife mit drei Etappen ergänzt. Als Warnung sei vorweggeschickt, dass im Folgenden auf den Unterschied zwischen *Standard* und *Spezifikation* weniger Wert gelegt wird, als dies in der einschlägigen Literatur der Fall zu sein pflegt.

PLAY [>] – Sommer 1996

Wir schreiben das Jahr 1 nach Internet Explorer. In den „Unix-Wars“ wird das Kriegsbeil zwischen der *Open Software Foundation (OSF)* und *Unix International* begraben und der Waffenstillstand durch den gemeinsamen Beitritt zur *X/Open Group* besiegelt. An der Linux-Front sind ein Quantensprung zur Versionsnummer 2.0, eine Pinguininvasion und stolze 0.5 Prozent Marktanteil im Serverbereich zu vermelden.

Das Ausmaß der durch das WWW hervorgerufenen Umwälzungen wird in ersten Konturen sichtbar, besonders für Beobachter auf der Empore. Der Historiker

Cargill, Spezialist für IT-Standardisierung und seinerzeit Verantwortlicher für dieses Gebiet bei der Firma Netscape, hat einen solchen Platz inne. Im Juli 1996 gelangt er (im bereits erwähnten Artikel) zur Einschätzung, dass das problematische Gegenteil zu *offen* nicht *proprietär*, sondern *geschlossen* ist. Diese Aussage mag sprachlich gesehen trivial erscheinen, ihre ökonomischen Konsequenzen waren und sind es nicht. Cargill zufolge war es insbesondere die im Vergleich zum damaligen Hauptkonkurrenten Apple offene Softwarearchitektur, die es Microsoft ermöglichte, seinen Mitbewerber zu deklassieren – ein aus heutiger Sicht bemerkenswertes Urteil.

Der Marktanteil des Netscape-Browsers beträgt zu jener Zeit stolze 90%, doch die Auslieferung der mit Windows 95 gebündelten Version des Internet Explorer 3 ist gerade noch einen Monat entfernt. In zwei Monaten wird Netscape eine Beschwerde bei der US-Justizbehörde einlegen. Das durch die maßgebliche Initiative von Tim Berners-Lee ins Leben gerufene *World Wide Web Consortium (W3C)* kämpft unterdessen mit den durch die Browser-Hersteller ohne Rücksprache eingeführten neuen HTML-Features. Die Nagelprobe für Standards ist, wie Cargill bemerkt, nicht die Art ihres Zustandekommens, sondern ihr Erfolg im Markt.

Am *Massachusetts Institute of Technology (MIT)* untersucht Doktorand Christopher Kelty die Auswirkungen des Internet auf die künftige IT-Infrastruktur und Standardisierung am Beispiel des Gesundheitswesens in den USA. Unterdessen brütet ein Werkstudent im Forschungslabor eines großen IT-Herstellers über ersten Entwürfen des „nächsten großen Dings“: *Secure Electronic Transactions (SET)*, einem umfangreichen Standard für sichere Finanztransaktionen in unsicheren Netzen. Denn soviel ist allen klar: Im Netz wird Geld fließen. Und SET, von Kreditkartenunternehmen unter Rückgriff auf offizielle Standards der *International Telecommunication Union (ITU)* spezifiziert und veröffentlicht, gilt als heißer Kandidat.

FF [>>] – Sommer 2000

Das Wissenschaftskomitee des US-Kongresses hat durch seinen Unterausschuss für Technologie ein Hearing zur heutigen und künftigen Rolle technischer Standards in der Gesellschaft anberaumen lassen. Eine der vorgetragenen Expertenmeinungen wird von der *Open Group* als Presseerklärung unter dem HTML-Titel „The Open Source Movement: Standardization Revisited“ im Web veröffentlicht werden (Cargill 2000). Hierin wird auf den wenig beachteten Umstand aufmerksam gemacht, dass nicht nur Standards selbst, sondern auch die für Standardisierung gewählten Organisationsformen einem evolutionären Prozess unterliegen.

Die Analyse kommt zu dem Schluss, dass Open Source als aktuellstes Ergebnis dieses Evolutionsprozesses interpretiert werden kann. Begründet wird dies damit, dass Open Source all jene Grundwerte beinhalte, die für jedes Standardisierungsmodell zuträfen: Befolgen von Spielregeln, Verfolgen eines gemeinsamen Interesse, Wunsch nach Veränderung, Planung und Leitung des Marktes und Legitimation durch die Vielfalt der Stakeholder.

Die Angemessenheit dieser Werte lässt sich nur vor dem Hintergrund eines ökonomischen Modells beurteilen, das die Untersuchung wie folgt benennt. Koopera-

tivem Verhalten von Konkurrenten schlägt für gewöhnlich weniger Misstrauen entgegen als den Aktivitäten eines Einzelanbieters; monopolistische Praktiken einzelner Anbieter werden vom Markt nicht, oder zumindest nicht über längere Zeiträume hin, geduldet. Es wird nicht näher auf die Frage eingegangen, ob dieses Modell der Logik des Marktes und der Open-Source-Entwicklung gleichermaßen gerecht wird; der Untersuchung geht es um das große Bild, um Zusammenschau:

> A group of well intentioned people, in an organization, can accomplish more in a more trusted manner than any individual acting alone, no matter how well intentioned or competent the individual is.

Diese Formulierung taucht, nahezu wortgleich, an zwei Stellen des Dokuments auf: als Schlüsselthema von Standardisierung und als essentielle Tugend von Open-Source.

PAUSE [| |]

Dem Organisationstyp von Open-Source-Entwicklung wird damit eine Rolle im Prozess der Standardisierung zugesprochen, die derjenigen von Allianzen, Konsortia, formalen nationalen und internationalen Institutionen oder Handelsvereinigungen vergleichbar ist. Das Zusammenwirken dieser Organisationstypen kann dabei in einer dreistufigen Sequenz gefasst werden.

Die Brown'sche Molekularbewegung des Marktes gebiert (1) implizite Standards in Form von Produkten. Wachsende Forderungen nach Kompatibilität und Interoperabilität bringen Konkurrenten (2) dazu, sich auf Industriestandards zu einigen, die *de facto* wirken. Diese werden (3), ausreichende Stabilität und Bedeutsamkeit vorausgesetzt, an formal verfasste Organisationen weitergereicht. Letztere verschaffen den jeweiligen Standards quasi-legalen Status und sichern ihnen einen Platz im institutionellen Gedächtnis – zumindest solange, bis der Markt diesen Standard für eine neue Generation von Produkten zu modifizieren und der Zyklus von vorn beginnt.

Innovativen organisatorischen Ansätzen für Standardisierung kommt in diesem Prozess eine wichtige Bedeutung zu. Es geht nicht allein um die Fähigkeit, für ein bestimmtes Standardisierungsvorhaben den geeigneten Organisationsrahmen zu wählen, sondern darum, bei Bedarf neue Organisationstypen zu kreieren. Europäische Unternehmen, so die Einschätzung des Papiers, seinen hier durch ihre Fixierung auf offizielle Standardisierung gehandicapt. In der Regel seien sie Unwillens, selbst informelle Gremien zu kreieren, drängelten sich allerdings um nachträgliche Aufnahme, sobald sich derartige Zusammenschlüsse anderenorts konstituierten.

Ein Grund für die Attraktivität flexibler, informeller Standardisierungsgruppen liegt in einem unterschiedlichen Verständnis dessen, was durch formal verfasste, internationale Gremien geregelt werden soll. Ihr (auch in den USA unangefochtener) Status als halb-öffentliches Gut macht Standards zu einem möglichen Vehikel für Governance, für Politik mit anderen Mitteln. Der Bericht konstatiert, dass IT-Standards in den USA primär technischer Natur seien, während in Europa zunehmend

nicht-technische Kriterien einbezogen würden, insbesondere Qualität, Sicherheit und Datenschutz.

Auch an anderen Stellen erweisen sich die Übergänge zum legalen und gesellschaftlichen Kontext als problematisch. So beklagt der Bericht die Unangemessenheit einer primär an Patenten orientierten Standardisierungspolitik für Internet-Technologien. In der erwähnten Anhörung gibt Martin Libicki, Policy Analyst bei der US-amerikanischen Denkfabrik RAND, Besorgnissen über eine zu laxe Vergabe von Softwarepatenten Ausdruck. Zu befürchten sind ihm zufolge negative Auswirkungen auf die bisherige Möglichkeit der lizenzfreien Nutzung von Standards (Libicki 2000).

CONT [>]

Dem Autor des erstzitierten Berichts (Cargill 2000) kommt auch in diesem Fall ein für Feldforschung idealer Aussichtspunkt zustatten. Cargill, mittlerweile Direktor für Firmenstandards bei SUN Microsystems, Inc. befindet sich im Zentrum des Zyklons, dem „dot in dot.com". Berufsbedingt ist er mit einer komplizierten Gratwanderung befasst: Der Standardisierung von Java.

Dieses aus Programmiersprache, virtueller Maschine und Schnittstellen bestehende Konstrukt ist geistiges Eigentum eines einzelnen Unternehmens, dem allerdings zur geordneten Ausbeutung seiner Entwicklung keine Atempause bleibt. Stattdessen liegt man im gerichtlichen Clinch mit einem großen Softwarekonzern aus Redmond. Streitpunkt: Das unabgesprochene Einfügen proprietärer Features und der damit einhergehende Missbrauch des Markenzeichens „Java compatible". Um die Spezifikation gegen unkontrollierbare Einflussnahme zu schützen, ist Java auf den langen Marsch durch die offiziellen Standardisierungsinstitutionen geschickt worden. Sollten damit Hoffnungen verbundene gewesen sein, auf diese Weise Zeit zu gewinnen und die eigene Position zu stabilisieren, so haben sich diese allerdings nicht erfüllt. Die möglichst rasche und weite Verbreitung von Java ist mehr denn je Gebot der Stunde, denn soeben ist mit C# und .NET eine Konkurrenzarchitektur aufgetaucht.

Im Bereich der Serverbetriebssysteme hat Linux inzwischen Novell hinter sich gelassen und liegt hinter NT auf Platz 2. Ein Gespenst geht um in der Entwicklergemeinde und bei jenen Pionieren, die Linux bereits im kommerziellen Umfeld einsetzen – das Gespenst der Fragmentierung. „Was würde passieren, wenn Linus morgen von einem Bus überfahren wird?" so die häufig gestellte Frage. Die *Linux Standard Base (LSB)* wird aus diesem Geist der Furcht, der Unsicherheit und des Zweifels geboren und schafft sich in Form der *Free Standards Group* ihren Gralshüter.

Das W3C (Eintrittspreis: 50000 Dollar p.a.) läuft der *Internet Engineering Task Force* (*IETF* – Eintritt frei) in punkto Relevanz zunehmend den Rang ab. Nur wenige bemerken bereits zu diesem Zeitpunkt, dass das W3C auf einer Sprengladung sitzt, an die bereits die Lunte gelegt worden ist: Die Präambel der Spezifikation für das *Simple Object Access Protocol (SOAP) 1.1* enthält inkompatible Auffassungen der Stakeholder über den Status des von ihnen beigesteuerten geistigen Eigentums.

Angetrieben von Kapitaleinlagen zu spät gekommener Gäste, die nicht wahrhaben wollen, dass die Zeit des Gauklers vorbei und die Party zuende ist, kämpft der NASDAQ in einem letzten, verzweifelten Klimmzug, bevor er seine Sturzfahrt in die Tiefe antritt. Am MIT hat Christopher Kelty seine Doktorarbeit „Scale and Convention: Programmed Languages in a Regulated America" verteidigt. Unterdessen brütet ein Ingenieur im Forschungslabor eines großen IT-Unternehmens über ersten Entwürfen des „nächsten großen Dings": TCPA, einer umfangreichen Spezifikation für die Operationalisierung multilateralen Vertrauens zwischen Endsystemen im Internet. Denn soviel ist allen klar: im Internet ist Vertrauen ein knappes Gut, besonders dann, wenn der Austausch von Information mit dem Austausch von ökonomisch messbaren Werten gekoppelt wird. Und TCPA, unter Federführung der wichtigsten IT-Unternehmen entwickelt, scheint das Zeug dazu zu haben, jenen Anteil des Transaktionsrisikos in den Griff zu bekommen, der durch (Fehl-)Konfiguration der Endsysteme zustande kommt. SET wurde in der Zwischenzeit spezifiziert, validiert, implementiert – und ignoriert. Es ist still geworden um diese Spezifikation. Vermisst wird sie nicht. Auch ohne dieses Protokoll hat die Anzahl der Geldüberweisungen im Internet explosionsartig zugenommen.

FF [>>] – Sommer 2003

Im Vorfeld der von seinem Verlag veranstalteten „Open Source Convention 2003" weist Tim O'Reilly in einem Interview darauf hin, dass sich auf dem Markt für Software derzeit Ähnliches abspiele wie ein Jahrzehnt zuvor auf dem Markt für PCs, es sei das selbe Muster, nur auf einer anderen Ebene. Software würde zunehmend in ihrer Eigenschaft als Komponente betrachtet, als Baustein, der sich mit anderen Bausteinen zu verteilten Diensten zusammensetzen lässt. Sie sei auf bestem Wege, zum Massengut zu werden; dieser Prozess werde durch Open Source und offene Standards vorangetrieben (O'Reilly 2003a). Die neuen Services seien jedoch weniger durch die sie konstituierende Software definiert als durch die strukturierten Daten, die durch ihre Interfaces wandern. Dieser Bereich sei, im Gegensatz zur durch Copyright und -left absicherbaren Software, schlecht geschützt und erfordere möglicherweise eine Art neuer „Bill of Rights" (O'Reilly 2003b).

Als entscheidendes Element für den Erfolg der Services benennt O'Reilly den offenen Charakter der Gemeinschaften, die sich um sie bilden. Dies ist ein Kerngedanke seiner Vision einer „Architektur der Partizipation" (O'Reilly 2003c). O'Reilly kritisiert den durch übermäßige Fixierung auf Softwarelizenzen beschränkten Horizont vieler Open-Source-Entwickler, der den Blick auf weitere wichtige Co-Faktoren erfolgreicher Innovation beinträchtige. Innovation komme im 21. Jahrhundert durch Überschneidung von offenen Protokollen, Gemeinschaften und Programmsourcen zustande.

Die Frage, ob diese drei Elemente in einer Rangbeziehung stehen, ist unter der Überschrift „Open Source vs. offene Standards" Thema eines aktuellen Disputs geworden (Schwartz 2003, Saint-Andre 2003). Die im Mai 2003 beendete langjährige Auseinandersetzung um die Patentpolitik des W3C[1] zeigt, dass es sich bei dieser

[1] Die öffentlicheWebsite der W3C Patent Policy Working Group findet sich unter der URL:

Diskussion um alles andere als ein belangloses Scharmützel handelt. Im Verlauf des Konflikts hatte sich insbesondere das Open-Source-Spektrum gegen eine Regelung ausgesprochen, die Elemente von Kommunikationsprotokollen unter den Schutz von „reasonable and non-discriminatory licensing" zu stellen erlaubte. Diese Form der Lizenzierung gestattet Inhabern von Patenten, Lizenzgebühren einzufordern. Die Gegner der Regelung bestanden dagegen auf prinzipieller Lizenzkostenfreiheit aller vom W3C abgesegneten Standards.

Die nun verabschiedete *Patent Policy* des W3C ist eine Kompromisslösung, die von den Gegnern von „reasonable and non-discriminatory licensing"-Klauseln als Sieg nach Punkten verbucht werden kann. Lizenzfreiheit ist der Regelfall, alle am Standardisierungsprozess Beteiligten müssen sämtliche relevanten eigenen Patente offenlegen und sind dazu angehalten, auf anwendbare Patente dritter Parteien hinzuweisen. Bei Ausnahmefällen bestimmen detaillierte Regeln, unter welchen Umständen eine patentgeschützte Technik in einen Standard mit aufgenommen werden kann.

Die Auseinandersetzung um die Patentpolitik des W3C zeichnete sich durch die ins Verfahren eingebauten Partizipations- und Konsultationsmechanismen sowie eine vorbildliche Dokumentation des Diskussionsstandes aus. Diese Mechanismen zur proaktiven Einbindung aller interessierten Parteien können als wegweisend angesehen werden, denn in Ermangelung derartiger Verfahrensregeln und Hilfsmittel schafft sich Öffentlichkeit gegebenenfalls auf andere Weise Gehör.

Hierfür legt die um TCPA entbrannte Kontroverse ein beredtes Zeugnis ab. Sie zeigt nicht zuletzt, dass die Veröffentlichung und freie Zugänglichkeit einer Spezifikation allein als legitimatorische Basis im Zweifelsfall nicht ausreicht. Die im Konsortium mitwirkenden Unternehmen wurden von einer quasi aus dem Nichts losgetretenen Debatte weitestgehend überrascht. Sie sahen sich zunächst vor der Schwierigkeit, den Ort zu lokalisieren, an dem sich die Auseinandersetzung abspielte. Besser gesagt *die Orte*, denn derer waren viele: Tagespresse, wissenschaftliche Zirkel, Fachpublikationen, eine schier unüberschaubare Vielzahl von Internetforen, Cyberaktivisten, politische Administration und Entscheidungsträger. Vorläufigen Höhepunkt bildet eine Anfang Juli 2003 zum Thema „Trusted Computing" einberufene, zweitägige Anhörung im Bundeswirtschaftministerium. Kein ungeeignetes Forum, da viele der im Zusammenhang mit „Trusted Computing" geäußerten Bedenken wettbewerbsrechtlicher Art sind.

REC [*]

Um welche Art von Offenheit geht es in diesen Auseinandersetzungen? Wie viele dieser Arten gibt es, und welche Rolle spielen sie in ihrem jeweiligen Kontext? Ergänzen sie sich oder können sie zu Widersprüchen führen?

Während sich im Fall von Sourcecode die verschiedenen Ansichten über das, was *offenen* Code ausmacht, in unterschiedlichen Copyright-Lizenzen (GPL, BSD, MIT, ...) niedergeschlagen haben, steht eine entsprechende Klärung im Bereich von Standards aus. Die von offiziell zuständigen Institutionen wie ISO, ITU oder ANSI

http://www.w3.org/2001/ppwg/.

verwendete Definition umfasst gerade einmal drei Kriterien: Offenheit für Stakeholder; Konsens; geregelte Verfahren für Konsultation und Einspruch.[2] Krechmer (1998) zählt eine Reihe weiterer Merkmale für Offenheit auf, die besonders bei der Standardisierung von IT bedeutsam, doch für die genannten Institutionen nicht verbindlich sind: Offenlegung von Patentansprüchen, weltweite Gültigkeit, freie Zugänglichkeit zu allen Dokumenten des Komitees und zu Versammlungen, Pflege von Standards, offene, d.h., erweiterungsfähige Interfaces, marginale Lizenzgebühren bzw. Lizenzfreiheit für Implementation akkreditierter Standards.

Nun ist es gerade nicht mangelnde Offenheit, die formalen Standardisierungsprozessen typischerweise angekreidet wird. Es waren vielmehr Vorwürfe der Langwierigkeit, Bindung an nationale politische Vorgaben und Marktferne, die zum Aufkommen leichtgewichtigerer Organisationstypen für Standardisierung im IT-Bereich führten. Paradoxerweise liegt ein Grund für die beklagte Schwerfälligkeit der traditionellen Gremien in formalen Verfahrensvorschriften zur Sicherstellung der offenen Meinungsbildung, nämlich dem Konsensprinzip und dem Procedere für Konsultation und Einspruch.

Obwohl weder die IETF noch das W3C die etwa für das ANSI gültigen Kriterien für Konsens und geregeltes Verfahren erfüllen (a.a.O.), kann deren Vorgehensweise – Zugang zu allen Drafts und Arbeitsdokumenten, kostenfreie Verfügbarkeit des Standardwortlauts – als Pluspunkt verbucht werden. Zumindest in den Augen vieler „Netizens" steht die Legitimität der von diesen beiden Organisationen verabschiedeten Spezifikationen derjenigen „offizieller" Standards in nichts nach. Doch auch bei Industriekonsortien, die nach landläufiger Meinung geschlossen und undemokratisch operieren, lohnt sich ein genaueres Hinschauen. Tineke Egyedi führt in ihrer Studie „Beyond Consortia, Beyond Standardization" eine Reihe empirischer Belege dafür an, dass die Offenheit solcher Konsortien gemeinhin unterschätzt, diejenige formaler Institutionen hingegen überschätzt wird. Ihre Untersuchungen weisen darauf hin, dass formale und informelle Organisationen auf sehr ähnliche Weise operieren und gleichermaßen auf Konsens und Berücksichtigung von Minderheitenmeinungen Wert legen (Egyedi 2001, S. 4–5).

Die in Open Source gesetzten Hoffnungen begründen sich aus diesem Grund weniger in ihrem partizipativen Modell *per se* als in ihrer Rolle als Korrektiv fehlender marktwirtschaftlicher Anreize (die Erstellung kompatibler IT-Lösungen kommt nämlich aus Sicht der Produzenten oft weniger ihnen selbst als vielmehr der Verbraucherseite zugute). Open-Source-Software (OSS) wird deshalb, wie Egyedi nahelegt, als Instrument zum Schutz und zur Durchsetzung von Verbraucherinteressen interessant. Diesem Gedanken mag die Beobachtung zugrundeliegen, dass Entwicklung von OSS oft unter Bedingungen erfolgt, die frei vom unmittelbaren Druck sind, konkurrierende Anbieter kommerziell zu übertrumpfen. Bedingungen mithin, die eine von Kartellrechtlern oft als *ultima ratio* verklärte Logik des marktwirtschaftlichen Wettbewerbs zum Teil ausser Kraft setzen. Dies fördert tendenziell jenes Maß an Kooperation zwischen Entwicklern, das nötig ist, um Code auf verschiedenen Plattformen zum Laufen zu bringen. In der Folge kommt es zu verbesserter Inter-

[2] Siehe FN 1.

operabilität, Kompatibilität und Portabilität sowie einer Tendenz zur Anwendung von Standards – selbst dann, wenn Standardkonformität nicht das Hauptziel der Entwicklungstätigkeit ist (Tze Meng 2003).

In eine ähnliche Kerbe schlägt Bruce Perens, dessen begriffliche Fassung von OSS als „nicht-konkurrierendem öffentlichen Gut" allerdings deren direkte marktwirtschaftliche Effekte genauer in den Blick nimmt. Perens steht derzeit vor der interessanten Herausforderung, die OSS-Strategie der Open Group voranzutreiben, also jener Organisation, die Offenheit als Markenzeichen sozusagen gepachtet (Perens 2003), in der Vergangenheit jedoch oft nur unzureichend umgesetzt hat. Perens ist sich der Ironie seines Unternehmens natürlich bewusst. Kritik und Häme der Open-Source-Gemeinde haben nicht lange auf sich warten lassen.[3] Der bisweilen genüsslich vergossene Spott schafft jedoch nicht die Tatsache aus der Welt, dass ein Teil der Open-Source-Entwicklung bereits seit geraumer Zeit von der beträchtlichen Erfahrung der Open Group in Sachen Spezifikation, Validierung und Tests profitiert. Als Beispiel bietet sich die bereits erwähnte *Linux Standard Base (LSB)* an, eine konzertierte Aktion zur Vereinheitlichung zentraler Komponenten von Linux-Distributionen. Im Auftrag der LSB-Gralshüter, die vielfach in der Wolle gefärbte Open-Source-Entwickler sind, definiert und managt die Open Group seit Mitte 2002 das hierfür notwendige Zertifikationsverfahren (Open Group 2002).

Der schiere Umfang ihrer jüngsten Aktivitäten (Open Group 2003) mag als Indiz gewertet werden, dass die Open Group ernsthafte programmatische Anstrengungen macht, das Paradigma der OSS-Entwicklung aufzugreifen. Die praktischen Ergebnisse dieser Bemühungen bleiben, nicht zuletzt vor dem Hintergrund historischer Erfahrungen, abzuwarten. Perens' Entwurf dokumentiert die zahlreichen Hürden, die es zu nehmen gilt. Dessen ungeachtet ist es der bislang wohl ambitionierteste Versuch, die Komplementarität von Open Source und offenen Standards in einem durch Wettbewerb geprägten Umfeld und „am lebenden Patienten" zu testen.

Selbst Totgeglaubten haucht die Aussicht auf gegenseitige Befeuerung von offenem Code und offenen Standards neuen Odem ein. Mit einiger Verwunderung beobachten wir derzeit eine fröhliche Urständ der öffentlichen Beschaffungspolitik als Mittel der Markt- und Technologiegestaltung.[4] Hierbei scheint es sich um ein internationales Phänomen zu handeln, das im Sinne Cargills, O'Reillys und Perens' gedeutet werden kann.

Solange Hardware- und Softwarefeatures die dominanten Marktfaktoren sind, ist die öffentliche Verwaltung auf die Position eines Kunden festgelegt, da sie typischerweise weder IT-Technologie noch Software produziert. Während der vergangenen Jahrzehnte ist der Einfluss der öffentlichen Hand auf die Gestaltung von Hard- und Software in eben jenem Maße gesunken, in dem diese zu Massengütern wurden, zumal maßgeschneiderte Systeme schon aus Kostengründen nicht mit kommerziellen „off-the-shelf"-Lösungen konkurrieren konnten.

---

3 Vgl. http://slashdot.org/articles/03/07/25/1517236.shtml sowie http://slashdot.org/comments.pl?sid=72427&cid=6533898.

4 Einen guten, international ausgerichteten Überblick bietet die Website des *Center of Open Source & Government* unter http://www.egovos.org/.

In jenem Maße jedoch, in dem sich das Marktgeschehen auf die höhere Ebene elektronischer Services verlagert, finden sich Verwaltungen in der Rolle von Service-Providern wieder, da sie Nettoproduzenten strukturierter und validierter Informationen sind. Hardware und Software als Massenartikel erlauben, diese Informationen nicht nur wie bisher für Zwecke interner Organisation, sondern auch für nach außen gerichtete Angebote einzusetzen. Als Konsequenz stellt sich z.B. die Frage offener Dokumentenformate in völlig neuem Lichte. Öffentliche Institutionen, die aus traditionellen und strukturellen Gründen seit jeher Befürworter offener Standards sind, haben ein schlagartig verdoppeltes Interesse an der Interoperabilität der Infrastruktur. Der so erzeugte Schub stellt Teile der IT-Industrie vor unerwartete Herausforderungen (Economist 2003).

REW [<<]

Der Kreis schließt sich, wir kehren zurück zum Anfang. Wie plausibel ist die eingangs dargestellte These der zirkulären Bewegung allen Standardisierens, der Wiederkehr des immer Gleichen? Und wie steht es um die Bewegung in der dritten Dimension, jener ins Offene?

Als vorläufiges Fazit können wir festhalten, dass diese Dimension möglicherweise nur in einer Projektion existiert. Offenheit weist eine Vielzahl von Dimensionen auf. Sie tritt uns als Attribut von Entscheidungen, Produktionsprozessen, Märkten, Sourcecode und Standards entgegen, um nur einige zu nennen.

Im Rückspiegel betrachtet, scheint die Gemeinsamkeit der verschiedenen Dimensionen von Offenheit in einer doppelten Funktion zu liegen: Jener der Legitimation und jener der Anschlussfähigkeit. Es ist jedoch nicht schwer, sich Situationen vorzustellen, in denen beide als Antagonismen auftreten, z.B. dort, wo die Grenzen zwischen Anschlussfähigkeit und Anschlusszwang verschwimmen.

Wo immer Code und Standards in gesellschaftlich normativer Weise wirken, wird unweigerlich nach deren Legitimation gefragt werden. Doch die Entscheidung über die Wirkungsmacht von Standards und Code erfolgt nicht an der Wahlurne und auch nicht im Gericht. Die Nagelprobe ist, wie Cargill bemerkt, pragmatischer Natur, denn die eigentliche Wirksamkeit von Standards manifestiert sich allein in ihrem aktiven Gebrauch. Dabei kann Anschlussfähigkeit zwar durch explizite Standardisierung befördert werden, setzt jene aber als nicht notwendig voraus. Anschlussfähigkeit ist im Prinzip bereits dann gegeben, wenn Bereitschaft dazu vorhanden ist, sich auf pragmatische und möglicherweise implizite Konventionen einzulassen.

Christopher Kelty würde als Voraussetzung für Anschlussfähigkeit vielleicht von der Bereitschaft sprechen, sich durch Sprache programmieren zu lassen. Seiner Arbeit[5] (Kelty 2000) verdanke ich die Einsicht, dass es nützlich ist, die Eigenschaften der jeweilig verwendeten Sprache in jenen Bereichen zu betrachten, die seit einiger Zeit unter dem Oberbegriff „Internet Governance“ zusammengefasst werden. Die

[5] Eine außergewöhnlich luzide Arbeit, deren Lektüre wärmstens ans Herz gelegt wird. Die URL im Literaturverzeichnis verweist auf das Inhaltsverzeichnis und die Einleitung. Der Gesamttext wird von C. Kelty auf Nachfrage zugänglich gemacht.

Frage der Offenheit scheint eng mit der Frage verknüpft zu sein, ob und wie Repräsentation reproduzierbar definiertes Verhalten oder Handlungen hervorbringt. Diese Beobachtung gilt gleichermaßen für Gesetze, Standards und Computerinstruktionen, das Stichwort hierzu lautet Performativität. Welche Sprache spricht der Markt?

Um die Erfolgsaussichten von Legitimationsversuchen, die mit einem partikularen Begriff des Offenen operieren und das Primat etwa des Politischen, Legalen, Ökonomischen oder Technischen zu behaupten versuchen, steht es jedenfalls nicht zum Besten. Was auch immer Offenheit sein mag, sie könnte, in Keltys Worten, auf dem Weg dazu sein, jenseits aller anderen Werte das einzige zu werden, das einen offenen Standard legitimiert. Der postmoderne Ausweg, also die Postulierung eines „Pluralismus der Offenheiten", scheint jedenfalls verbaut: Das Offene hat keinen Plural, sondern nur ein Gegenteil.

(*) Die in diesem Beitrag vertretenen Ansichten sind weder in Teilen noch als Ganzes notwendigerweise diejenigen meines Arbeitgebers.

Literatur

Cargill, Carl (1996): *The Rise and Fall and Rise of Standardization*, in: UniForum 07/1996, S. 2–3

Cargill, Carl (2000): *Evolutionary Pressures in Standardization: Considerations On Ansi's National Standards Strategy*, online http://www.house.gov/science/cargill_091300.htm

The Economist (2003): *Microsoft at the power point*, online http://www.economist.com/business/displayStory.cfm?story_id=2054746

Egyedi, Tineke (2001): *Beyond Consortia, Beyond Standardisation?*, Final Report for the European Commission, online http://europa.eu.int/comm/enterprise/standards_policy/study/consortia_standardisation/final_report_delft_en.pdf

Kelty, Christopher M. (2000): *Scale and Convention: Programmed Languages in a Regulated America*, PhD Thesis, MIT, online http://viz.kelty.org/thesis/index.html

Krechmer, Ken (1998): *The Principles of Open Standards*, in: Standards Engineering, Vol. 50, No. 6, S. 1–6, online http://www.csrstds.com/openstds.html

Libicki, Martin C. (2000): *The Role of Standards in Today's Society and in the Future*, online http://www.house.gov/science/libicki_091300.htm

The Open Group (2002): *The Free Standards Group Announces LSB Certification Program*, online http://www.opengroup.org/press/articles/02-06%20LSB%20Certification.pdf

The Open Group (2003): *Open Source in The Open Group*, online http://www.opengroup.org/tech/open-source/index.htm

O'Reilly, Tim (2003a): *Software licenses don't work*, Interview mit Robert McMillan, online http://infoworld.com/article/03/07/03/HNoreilly_1.html
O'Reilly, Tim (2003b) *Session: Bill of Rights for Web Services*, Open Source Convention, 10. Juli 2003, online http://conferences.oreillynet.com/cs/os2003/view/e_sess/4526
O'Reilly, Tim (2003c): *The Architecture of Participation*, online http://www.oreillynet.com/pub/wlg/3017
Perens, Bruce (2003): *An Open Source Strategy for the Open Group – Draft for Comments*, The Open Group, online http://www.opengroup.org/tech/open-source/opengroup-os-strategy.htm
Saint-Andre, Peter (2003): *Open Source and Open Standards*, online http://www.onlamp.com/pub/a/onlamp/2003/04/29/openstandardsopensource.html
Schwartz, Jonathan (2003): *Open source versus open standards*, online http://news.com.com/2010-1071-995823.html?tag=fd_nc_1
Tze Meng, Tan (2003): *The Case for Open Source: OSS vs. Proprietary Software*, online http://opensource.mimos.my/fosscon2003cd/paper/full_paper/tan_tze_meng.pdf

Erfolgsfaktoren bei der Einführung von Linux in Unternehmen

PETER H. GANTEN

1 Einleitung und Aufgabenstellung

Die Ausgangssituationen mittelständischer Unternehmen hinsichtlich der Einführung bzw. dem systematischen Ausbau der Verwendung von Open-Source-Software (OSS) können – auf den ersten Blick betrachtet – unterschiedlicher nicht sein. Allein für den Begriff „mittelständisches Unternehmen" gibt es unterschiedliche Definitionen. In diesem Artikel sollen darunter Unternehmen mit wenigstens 100 und höchstens 2000 Mitarbeitern verstanden werden. Es ist selbstverständlich, dass man für ein so breites Spektrum von Unternehmen aus ganz unterschiedlichen Branchen kein „Kochrezept" zur Einführung von Linux oder anderer OSS geben kann. Im Gegenteil: Abhängig von der technischen Ist-Situation, von den in einem bestimmten Unternehmen mit Hilfe der Informationstechnologie (IT) zu lösenden Problemen sowie den kurz-, mittel- und langfristigen Zielen, insbesondere aber auch von der Art und Menge technischer Kompetenz und nicht zuletzt abhängig vom Budget, können die konkreten Schritte bei der Einführung von OSS völlig unterschiedlich aussehen.

Zwei Beispiele seien hier genannt: Auf der einen Seite eine Regionalbank (ca. 2000 Mitarbeiter), welche die praktischen und strategischen Vorteile von OSS für sich erkannt hat, Linux bereits für eine Reihe von Aufgaben einsetzt und nun konkrete Hilfe bei der Optimierung eines Bereiches ihrer IT-Infrastruktur auf der Basis Linux benötigt. Und auf der anderen Seite eine mittelständische Spedition unter familiärer Führung (ca. 400 Mitarbeiter), die zunächst einmal nur erfahren möchte, ob und in welchen Bereichen der Einsatz von Linux und anderer OSS praktikabel und lohnenswert ist. Während es im ersten Beispiel zunächst so aussieht, als müssten die grundsätzlichen Fragen des Einsatzes von OSS dort nicht mehr beleuchtet werden und die Arbeit der externen und internen Fachleute dort zum großen Teil klassische IT-Konzeptionierungs- und -Implementierungstätigkeit wäre, muss im zweiten Fall (der Spedition) zunächst eine genaue Analyse der Ist-Situation hinsichtlich technischer Voraussetzungen und der Unternehmensstruktur sowie hinsichtlich der Ziele des Unternehmens stehen.

Nichtsdestotrotz gibt es viele Gemeinsamkeiten und letztendlich eine mögliche Basisstrategie, die in immer abgewandelter Form, mit unterschiedlichen Schwerpunkten und Erweiterungen, die Analyse der Potentiale von OSS für mittelständische Unternehmen sowie die erfolgreiche Einführung begleitet. Zwei der wichtigsten Komponenten dieser Basisstrategie sind – so profan sich das anhören mag – die Ist-Analyse bzgl. technischer, wirtschaftlicher und personeller Voraussetzungen und die gemeinsame Aufstellung von Zielen für die Weiterentwicklung der IT-Infra-

struktur. Nichts ist gefährlicher, als das „Drauf-los"-Implementieren technisch ausgezeichneter Lösungen, wenn sie nicht in die Gesamtstrategie des Unternehmens passen. Zwingende Voraussetzung, um dies zu vermeiden, ist natürlich das Vorhandensein einer Gesamtstrategie.

Die IT-Gesamtstrategie eines Unternehmens ist heute immer auch eine Open-Source-Strategie (OS-Strategie), selbst wenn sie im Hinblick auf OSS nur die Aussage macht, dass OSS nicht eingesetzt wird. Von Personen mit IT-Entscheidungsverantwortung kann heute erwartet werden, dass sie sich mit den Eigenschaften von OSS auseinandersetzen und diese in Relation zu den Zielen des eigenen Unternehmens setzen, um sich dann in verschiedenen Bereichen für oder gegen den Einsatz von OSS zu entscheiden. Eine solche Entscheidung ist selbst dann wichtig und sinnvoll, wenn heute noch keine für das betreffende Unternehmen einsetzbaren OSS-Lösungen zur Verfügung stehen. Denn auch, wenn zunächst proprietäre Software eingesetzt werden muss, können Auswahl und Einsatzart solcher Software von der grundsätzlichen Entscheidung abhängig sein und dem Unternehmen überhaupt erst die Möglichkeit eröffnen, zukünftig OSS einzusetzen.

Dieser Aufsatz ist folgendermaßen gegliedert: Zunächst soll die Entwicklung einer OS-Strategie in mittelständischen Unternehmen beschrieben werden. Davon ausgehend wird die Umsetzung einer möglichen OS-Strategie, nämlich Konzeptionierung und Durchführung einer zweistufigen Migration (zunächst der Server und dann der Clients) beschrieben.

2 Ist-Aufnahme und Entwicklung einer OS-Strategie

Die Entwicklung einer OS-Strategie beruht auf zwei Säulen: Den mittel- und langfristigen strategischen Zielen des betreffenden Unternehmens sowie der eingehenden Analyse der wirtschaftlichen und technischen Ist-Situation.

Die Analyse des Ist-Zustands dient zum einen als fundierte Datenbasis für die realistische Bestimmung von Zielen und Strategien, sie soll gleichzeitig auch ausreichend Informationen liefern, um die Einführung von Linux und anderer OSS in dafür als geeignet identifizierten Bereichen grob konzipieren und den Aufwand abschätzen zu können.

Am Anfang steht in der Regel ein Kick-Off-Meeting, an dem neben den eigentlich für die Migration verantwortlichen Personen (internes Personal und/oder externe Berater) die für den IT-Bereich verantwortlichen Mitarbeiter der Leitungsebene und die Administratoren der betreffenden Organisation teilnehmen. Ziele des Kick-Off-Meetings sind die Schaffung eines gemeinsamen, eindeutigen Verständnisses der Ziele für die Analysephase („Wir wollen herausfinden, in welchem Bereich der Einsatz von OSS für unser Unternehmen Sinn machen kann," und nicht: „Wir führen jetzt Linux ein und schauen, wie das am besten gehen kann.") sowie die Definition der einzubeziehenden Komponenten, falls hier eine Einschränkung notwendig ist. Die Vorstellung der Vorgehensweise bei der Analyse, die Grobkatalogisierung der IT-Komponenten und die Identifikation von Experten für bestimmte Teilbereiche, Anwendungen oder Server runden das Kick-Off-Meeting ab.

Die Analyse sollte mindestens die folgenden Teilbereiche umfassen, auf die im folgenden detailliert eingegangen wird:

- Kostensituation aus der Sicht des Managements;
- Administrativer Aufwand;
- Netzwerk- und Serverinfrastruktur (Hard- und Software);
- Clients (Hardware, Software, Arbeitsabläufe, Administration);
- Schnittstellen des Unternehmens / der Organisation;
- Vorhandene menschliche Ressourcen (Know-How, Lernbereitschaft);
- Vorhandene Konzepte (z.B. Betrieb, Backup, Sicherheit).

2.1 Kostensituation aus Sicht des Managements

Die Kostensituation wird in zwei Schritten untersucht. Einmal vor Durchführung der technischen Analyse auf Basis des von der Unternehmensleitung beschriebenen Zustands und, im anschließenden zweiten Durchgang, auf der Basis der technischen Analyseergebnisse. Gerade wenn die Leitung nicht genau weiss, was die IT-Infrastruktur eigentlich kostet, und in welche Bereiche sich die Kosten aufteilen, ist es wichtig, die folgenden Fragen vor der technischen Analyse zu stellen, weil dadurch ersichtlich wird, in welchen Bereichen verdeckte Kosten vorhanden sein können, die sich durch eine Optimierung der IT-Infrastruktur reduzieren lassen. Zusätzlich wird die Notwendigkeit der nachfolgenden technischen Analyse leichter nachvollziehbar, wenn beim ersten Durchgang der Kostenanalyse Fragen offen bleiben, die dann im zweiten Durchgang (nach der technischen Analyse) geklärt werden müssen.

Die folgenden zentralen Teile der IT-Kosten in Unternehmen müssen untersucht werden:

- Hardwarekosten (durchschnittliche Anschaffungskosten, Hardware-Support, Instandhaltung, Hardwareüberwachung etc.);
- Direkte Softwarekosten (durchschnittlicher Aufwand für Lizenzkosten, Aufwand für Softwarepflegeverträge, Aufwand für Erstellung und Pflege von Eigenentwicklungen, Aufwand für Softwareerstellung und -pflege durch Dritte);
- Indirekte Softwarekosten (Aufwände für Installation, Anpassung, Aktualisierung und Verteilung von Software, Aufwand für die Verwaltung von Lizenzen etc.);
- Administration und Help-Desk: Wieviele Administratoren, Help-Desk-Mitarbeiter beschäftigt das Unternehmen? Beschäftigen sich auch Personen mit der Administration, die eigentlich andere Aufgaben haben? Wieviel Arbeitszeit der Administratoren wird für administrative Aufgaben verwendet, wieviel für Unterstützung von Endbenutzern? Wieviel Arbeitszeit geht für welche administrativen Arbeiten verloren (Administration von Benutzern und Rechten, Backup und Restore von Daten, Aufsetzen von Clientsystemen, Installation, Konfiguration und Integration neuer Server, Softwareinstallation und -aktualisierung)?

  Wieviel Arbeitszeit verwenden Benutzer darauf, Ihren Arbeitsplatz kennenzulernen, sich einzurichten oder eigene Software zu installieren? Wieviel Ar-

beitszeit geht dadurch verloren, dass Benutzer anderen Benutzern Hilfestellung bei der Verwendung von Programmen geben?
- Kostenrisiken: In welchen Bereichen ist das Unternehmen Preiserhöhungen oder Änderungen der Zahlungsmodalitäten relativ schutzlos ausgeliefert? Welche Kosten können durch Ausnutzung von Sicherheitslöchern entstehen? Welche Risiken bestehen bei der Weiterentwicklung der Infrastruktur, beispielsweise durch Inkompatibilitäten eingesetzter Software mit neueren Betriebssystemversionen?
- Arbeitszeitverlust: In welchem Umfang sind in der Vergangenheit Arbeitszeitverluste durch Probleme (Ausfälle von Servern oder Clients, Viren, versehentliches Löschen von Daten etc.) entstanden? Hat die betreffende Organisation neben dem eigentlichen Verlust der Arbeitskraft dadurch weitere Verluste erlitten (z.B. verlorene Aufträge, verlorenes Kunden- / Partnervertrauen, nicht eingehaltene Liefertermine)?

2.2 Technische Analyse

Die technische Analyse untersucht alle Bestandteile der IT-Infrastruktur sowie die Art, in der diese von dem Unternehmen genutzt werden. Im Hinblick auf die mögliche Migration zu Linux und anderer OSS sind vor allem die Analyse von Server- und Client-Infrastruktur sowie deren Benutzung von zentraler Bedeutung.

Elemente der Analyse im Serverbereich sind die Erfassung aller Serversysteme und den darauf ausgeführten Diensten und Anwendungen sowie die Auslastung, Verfügbarkeit, Stabilität und Sicherheit der Server. Besonders wichtig sind die Identifikation von Interdependenzen zwischen Systemen, Diensten und Anwendungen. Welche Datenbanken werden von der Auftragsbearbeitung benötigt? Wogegen erfolgt die Authentifizierung von Benutzern am Content Management System? Schließlich muss untersucht werden, welche Dienste und Serveranwendungen von welchen Benutzern in welchem Umfang verwendet werden. Wir haben oftmals die Erfahrung gemacht, dass bestimmte Systeme gar nicht mehr oder nur von ein oder zwei Benutzern verwendet wurden und den Aufwand zur Bereitstellung und Pflege längst nicht mehr rechtfertigten.

Viele im Rahmen der Analyse im Serverbereich benötigten Informationen können durch Befragung der Administratoren gewonnen werden. Solche Befragungen müssen auch genutzt werden, um den Arbeitsalltag der Administratoren in dem betreffenden Unternehmen kennenzulernen. Dadurch kann am ehesten festgestellt werden, welche Bereiche der Administration besonders aufwändig sind und wo es regelmäßig zu Problemen kommen kann.

Im Bereich der Clients umfasst die Analyse die Aufnahme und grobe Kategorisierung der vorhandenen Client-Hardware, der auf den Clients verwendeten Betriebssysteme und -Versionen und der darauf installierten Anwendungen. In vielen Organisationen lassen sich Clients in unterschiedliche Kategorien einteilen (z.B. Vertrieb und Sachbearbeitung, Produktion und Management). Eine solche Einteilung kann hilfreich sein, um später Teilbereiche zu identifizieren, in denen der Einsatz bestimmter Lösungen besonders sinnvoll oder weniger sinnvoll ist. Zur Analyse der Clients gehört aber auch eine Befragung und Beobachtung typischer Benutzer

aus den einzelnen Bereichen. Die beste Vorgehensweise besteht darin, sich zwei Benutzer aus jedem Bereich zu suchen und sich von diesen „Ihren" Arbeitsplatz erklären zu lassen. Wie werden bestimmte Dinge bearbeitet? Welche Anwendungen und Vorgehensweisen werden dazu genutzt? Welchen Zweck haben die Icons in Menüs oder auf dem Desktop? Auf welche Server wird zugegriffen? Was gefällt dem betreffenden Mitarbeiter an seinem Arbeitsplatz besonders gut, was hat ihn schon immer gestört? Gerade die letzten beiden Fragen können helfen, die Akzeptanz neuer Lösungen zu erhöhen, wenn die Antworten beherzigt werden.

Benutzer und Administratoren sollten über Stabilität, Sicherheit und Verfügbarkeit der Clients befragt werden. Wie lange dauert das morgendliche Hochfahren des Rechners? Wie oft muss die Arbeit unterbrochen werden, weil Serversysteme nicht verfügbar sind? Gab es Arbeitsbehinderungen oder Datenverlust auf Grund von Viren?

Der letzte Fragenkomplex bei der Analyse der Arbeitsplatzsysteme betrifft deren Administration: Wie wird Software verteilt und ein einheitlicher Releasestand sichergestellt? Welche Methoden stehen zur Verfügung, um ein wichtiges, sicherheitsrelevantes Update in sehr kurzer Zeit einspielen zu können? Wie werden unterschiedliche Benutzerprofile verwaltet und gepflegt? In welchem Umfang dürfen Benutzer ihre Arbeitsplatzsysteme selbst administrieren?

Die erfolgreiche Client-Analyse macht also Aussagen über die verwendeten Anwendungen und Verfahren, aber auch über die auf dem Client vorhandenen, aber nie genutzten Programme, genauso wie über Wünsche der Mitarbeiter an die Gestaltung ihres Arbeitsplatzes. Natürlich sind auch rein serverseitig ausgeführte Anwendungen (wie z.B. Web-basierte Anwendungen) und die für die aktuelle Funktionsweise der relevanten Anwendungen benötigten Serverdienste mit zu erfassen.

Neben Servern und Clients müssen auch alle übrigen Komponenten der IT-Infrastruktur erfasst werden. Dazu gehören Drucker (wird lokal oder über Server gedruckt?), Notebooks (wie erfolgen Pflege, Datensynchronisation und Virenschutz?), Telearbeitsplätze (wie erfolgen Anbindung und Pflege?), sowie weitere Komponenten (z.B.: CD-Brenner, Diskettenlaufwerke, ISDN-Karten, Scanner oder PDAs).

Ein äußerst wichtiger Aspekt sind die Schnittstellen eines Unternehmens nach außen, also etwa zu Partnern und Kunden. Welche Dokumente werden in welchen Datenformaten ausgetauscht? Was geschieht nach dem Austausch mit den Daten? Gibt es Serverdienste, die bestimmte Daten für externe Benutzer bereitstellen, etwa eine XML-Schnittstelle, über die Kunden Katalogabfragen vornehmen können?

Die Analyse sollte außerdem ein ausreichend fundiertes Bild über das in der betreffenden Organisation vorhandene Know-How sowie mögliche Vorbehalte gegenüber Veränderungen liefern. Wie sehr beherrschen die Administratoren die zur Zeit eingesetzten Systeme? Sind sie mit den Systemen zufrieden und wo wünschen sie sich Verbesserungen? Gibt es Know-How bezüglich der Administration anderer Systeme (z.B. UNIX/Linux)? Und: Für wen könnte die Migration zu anderen Systemen eine Gefahr darstellen, etwa weil er oder sie dann nicht mehr der unangefochtene Experte für eine bestimmte Komponente ist?

Ein gutes Bild über das qualitative Niveau der Administration in einem Unternehmen oder einer Behörde ergibt sich aus der Durchsicht vorhandener Konzepte

und Dokumentationen. Gibt es Sicherheits- und Verfügbarkeitskonzepte, in denen die Abläufe beim Ausfall von Systemen beschrieben sind? Gibt es ein Betriebskonzept das alle oder viele Abläufe und Verfahren beschreibt? Wie werden die vorhandenen Konzepte „gelebt"?

Die Ergebnisse der Analyse müssen ausführlich dokumentiert und gut verständlich aufbereitet werden.

2.3 Kostensituation (zweiter Durchgang)

Mit den Daten der technischen Analyse kann die Kostensituation neu aufgeschlüsselt werden. Insbesondere die verdeckten Kosten wie Kosten durch Ausfallzeiten, Kosten administrativer Abläufe, Kosten durch Arbeitsabläufe der Mitarbeiter, Kosten durch aufgabenfremde Tätigkeiten von Mitarbeitern, Kosten durch Viren oder Kosten durch Abhängigkeit von bestimmten Softwarekomponenten können nun genauer beurteilt werden.

Nach Darstellung der Kostensituation und Gegenüberstellung mit den Daten des ersten Durchgangs werden die zentralen Punkte in einer Präsentation zusammengefasst. In einem zweiten Meeting, an dem dieselben Personen teilnehmen, die auch im Kick-Off-Meeting anwesend waren, werden die Ergebnisse vorgestellt. Ziele dieses Meetings sind die Herstellung eines ähnlichen Verständnisses der Teilnehmer von der Kostensituation der IT-Infrastruktur des Unternehmens, die gemeinsame Definition von kurz-, mittel- und langfristigen Zielen bei der Weiterentwicklung der IT-Infrastruktur, sowie die Definition der Grundlagen für die Entwicklung einer OS-Strategie.

2.4 Entwicklung einer OS-Strategie

Darstellung der Ist-Situation im technischen und im wirtschaftlichen Bereich, sowie die Ziele des betreffenden Unternehmens hinsichtlich der Optimierung und Weiterentwicklung dieser Situation bilden die Basis für die Entwicklung einer OS-Strategie. Bezüglich der kurz-, mittel- und langfristigen Ziele sind zunächst die folgenden Fragen zu klären:

- Ist das Unternehmen abhängig von bestimmten Software- oder Hardwareprodukten? Wie teuer ist diese Abhängigkeit auf Dauer für das Unternehmen?
- Ist der Schutz von Know-How für das Unternehmen wichtig? Oder kann es toleriert werden, dass Softwareanbieter immer wieder neue Konzepte und Werkzeuge liefern, weil diese ohnehin nur durch externe Dienstleister adaptiert werden?
- Benötigt das Unternehmen Investitionssicherheit bei der Wahl von IT-Plattformen? Müssen die eingesetzten Systeme auch nach vielen Jahren noch wart- und weiterentwickelbar sein?
- Ist es für das Unternehmen wichtig, auf die Gestaltung bestimmter Programme selbst Einfluss zu nehmen oder durch einen Dienstleister Einfluss nehmen zu lassen? (Beispiel: Auf Grund von immer wechselnden Kundenanforderungen muss es flexible Möglichkeiten zum Schaffen von Schnittstellen zu bestimmten Programmen geben.)

- Werden in dem Unternehmen sicherheitsrelevante Systeme eingesetzt, bei denen es unbedingt erforderlich ist, den eingesetzten Code jederzeit überprüfen oder überprüfen lassen zu können?
- Will das Unternehmen mehr Möglichkeiten bei der Auswahl von Hardware haben? (Beispiel: Auf dem Host-System sollen die gleichen Programme ausgeführt werden können, wie auf den Arbeitsplatzrechnern.)
- Sollen administrative Vorgänge – auch über Systemgrenzen hinweg – leicht automatisierbar sein?
- Will das Unternehmen den IT-Verantwortlichen die Möglichkeit geben, die eingesetzten Systeme nachhaltig zu verstehen? (Beispiel: Möglichkeit zur Abklärung von Fragen anhand des Programmquellcodes.)
- Möchte das Unternehmen seinen Mitarbeitern die Möglichkeit geben, die eingesetzten Standardorogramme ohne weitere Kosten auch zu Hause verwenden zu können?
- Müssen unterschiedliche Systeme verschiedener Hersteller miteinander verbunden werden? (Beispiel: Etablierung einer einheitlicher Administration von Systemauthentifizierung und der Authentifizierung an einer Groupware-Anwendung.)

Je eindeutiger und je häufiger diese Fragen mit „ja" beantwortet werden, desto mehr spricht für den Einsatz von Open-Source-Software. Demgegenüber können in manchen Bereichen auch gute Argumente für den Einsatz proprietärer Software stehen:

- Gute Erfahrungen mit einem bestimmten Produkt und dessen Hersteller.
- Durch das Unternehmen nicht zu vertretender Zwang, bestimmte Produkte einzusetzen. (Beispiel: Proprietäre Kommunikationssoftware, die auch von den Kunden des Unternehmens eingesetzt werden.)
- Unmöglicher Verzicht auf Software, für die es keinen sinnvollen freien Ersatz gibt.
- Preislicher Vorteil einer bestimmten proprietären Lösung gegenüber einer OSS-Eigenentwicklung oder aufwendigen Anpassung.

Nach der Beantwortung beider Fragekomplexe durch die IT-Verantwortlichen sollte man in der Lage sein, eine grundsätzliche Aussage zur OSS-Strategie zu machen, nämlich: „Ja – wir müssen verstärkt OSS einsetzen", oder: „Nein – der Einsatz von OSS macht in unserem Unternehmen keinen Sinn." Auch sollte es möglich sein, Bereiche der IT-Infrastruktur zur identifizieren, in denen die Verwendung von OSS aus strategischen oder Kostengesichtspunkten dringlicher erscheint als in anderen.

Entscheidet sich ein Unternehmen für den Einsatz von OSS, dann müssen Ideen zur Grobkonzeption der Umsetzung dieser Entscheidung gesammelt werden. Von der sofortigen Ablösung bestimmter Programme oder Betriebssysteme bis zum mittelfristigen Verzicht auf die Verwendung ausgewählter Dateiformate, die mit OSS voraussichtlich Probleme bereiten, sind hier in unterschiedlichen Situationen ganz unterschiedliche Vorgehensweisen probat. Entscheidend sind dabei meist die

in der Regel eher positiven, in bestimmten Fällen aber möglicherweise auch negativen Konsequenzen der OSS-Einführung.

Eine relativ vollständige OSS-Strategie beinhaltet folgende Aussagen:

- Ob und warum das betreffende Unternehmen OSS einsetzen will.
- In welchen Bereichen (Sicherheit, Server, Clients) der Einsatz von OSS am dringlichsten ist und warum.
- Welche Verbesserungen sich durch den Einsatz von OSS ergeben sollen und wie das Erreichen dieser Verbesserungen durch andere Maßnahmen flankiert werden muss.
- Wie die Wirtschaftlichkeit der IT-Infrastruktur durch OSS gesteigert werden soll. Dabei sind neben den vorhersagbaren Kosten auch Risiken (beispielsweise durch Viren, Abstürze oder Angriffe von außen) mit einzubeziehen und geeignet zu bewerten.
- Maßnahmenkatalog mit Zeitpunkten.
- Budgetierung: Welches Budget kann sinnvollerweise für die Migration nach OSS bzw. für deren Ausbau aufgewandt werden? Wie ist das Budget zu verteilen (Mitarbeiterqualifizierung, Durchführung eigener Projekte, externe Dienstleister, Kauf von Lizenzen für proprietäre Software)?

Letztendlich wird man sich in den seltensten Fällen für eine sofortige komplette Ablösung proprietärer Software durch Freie Software entscheiden, sondern Mischformen bei der Weiterentwicklung der Infrastruktur zwischen fortführender und ablösender Migration finden. In bestimmten Bereichen kann sogar die Neubeschaffung proprietärer Produkte Sinn machen, etwa, weil diese den Einsatz von OSS für das Unternehmen überhaupt erst sinnvoll möglich machen. Wichtig ist es, dass in OS-Strategien keine „Hausaufgaben" versteckt werden, die das Unternehmen sowieso schon lange hätte erledigen müssen, wie beispielsweise das Anfertigen von Sicherheits- oder Betriebskonzepten. Außerdem sollten die Kosten der Einführung von OSS den Kosten der fortführenden Migration vorhandener, proprietärer Systeme gegenübergestellt werden, selbstverständlich unter Einbeziehung aller bei der Analyse bekannt gewordenen verdeckten Kosten.

Getrennt werden muss zwischen Servern und Clients. Im Serverbereich sind OSS-Migrationen in vielen Fällen einfacher durchzuführen als auf der Seite der Clients – dafür ist das potential zur Steigerung der Wirtschaftlichkeit bei Verwendung von OSS auf den Clients oft erheblich höher. Im folgenden sollen deswegen die Strategien und Optionen bei der Migration von Servern und Clients getrennt beschrieben werden.

3 Migration der Serverinfrastrukturen

3.1 File-, Print- und Authentifizierungsdienste, Verzeichnisdienste

Die Migration Windows-NT-basierter Serverinfrastrukturen nach Linux ist seit der Existenz von Samba 3.x relativ unproblematisch. Samba ist eine freie Software, die sich aus der Sicht eines Windows-basierten Systems wie ein Windows-Server verhält und die gleichen Dienste, nämlich File-, Print- und Authentifizierungsservices bereitstellt. Samba 3.x hat im wesentlichen denselben Funktionsumfang,

wie ein Windows-basierter Domänencontroller und unterstützt die Migration von Windows NT nach Linux/Samba durch die Möglichkeit, die Benutzer-, Gruppen- und Computerdatenbank eines Windows-NT-basierten Domänencontrollers zu übernehmen.

Die Tatsache, dass Samba sich gegenüber Windows-Systemen wie ein NT-basierter Domänencontroller verhält, bedeutet allerdings nicht, dass das Programm genauso implementiert ist. Viele Einschränkungen, die für Windows NT gelten (wie beispielsweise die beschränkte Größe der Benutzerdatenbank) gelten für Samba nicht. So stellt Samba eine ganze Reihe unterschiedlicher Methoden für die Datenhaltung der Benutzerdatenbank zur Verfügung. Ein für komplexere Infrastrukturen sicherlich interessantes Verfahren zur Speicherung der Benutzerdatenbank ist die Verwendung eines LDAP-kompatiblen Verzeichnisdienstes wie des freien und für Linux verfügbaren OpenLDAP:

Ein Verzeichnisdienst ist eine Art Datenbank mit hierarchischer Strukturierung. Dadurch eignen sich Verzeichnisdienste besonders gut zur Abbildung der internen Strukturen von Organisationen. Die hierarchische Strukturierbarkeit und hohe Flexibilität sowie die Möglichkeit zur Verteilung des Verzeichnisses auf viele Server und Standorte (Replikation und Partitionierung) machen Verzeichnisdienste heute zur oft idealen Antwort auf die komplexen Bedürfnisse von Organisationen hinsichtlich der Administration von Benutzern, Gruppen, Computern, Konfigurationen und beliebigen anderen Daten. Nicht zuletzt deswegen hat auch Microsoft die Nachfolgeprodukte von Windows NT Server (Windows 2000/2003 Server) mit einem Verzeichnisdienst ausgestattet. Auch andere Anbieter (Novell, Sun, IBM) setzen Verzeichnisdienste als integrale Bestandteile komplexer Infrastrukturen ein.

Dieser Trend hat erfreulicherweise dazu geführt, dass es auch immer mehr Anwendungen und Dienste unterstützen, Informationen über Benutzer, Gruppen, deren Rechte und Eigenschaften sowie andere Konfigurationsdaten aus Verzeichnisdiensten zu lesen. Für Organisationen bietet sich damit die Chance, durch den Einsatz eines Verzeichnisdienstes mittelfristig einen „Single-Point-of-Administration" zu realisieren, also die Administration zu weiten Teilen ausschließlich mit dem Verzeichnis durchzuführen und beispielsweise darauf verzichten zu können, neue Benutzer regelmäßig in den unterschiedlichsten Systemen (z.B. Windows-Domäne, SAP, Proxy, Firewall, Content-Management-System, usw.) anlegen zu müssen, bzw. die Konten in allen Systemen aktuell und synchron zu halten.

Auf der anderen Seite bringt die Einführung eines Verzeichnisdienstes einen gewissen planerischen Aufwand mit sich. Nicht zuletzt deswegen sind insbesondere Organisation, die über eine Windows-NT-Domäne verfügen, gut beraten Linux, OpenLDAP und Samba als Alternative zu einzusetzen, ernsthaft zu prüfen. Der Umstellungs- und Lernaufwand ist vergleichbar, die Migration nach Samba ist oft sogar einfacher, weil dort aus Sicht der Windows-basierten Client-Systeme alte Konzepte beibehalten werden. So kann ein Samba-System sogar so eingerichtet werden, dass die traditionell unter Windows NT verwendeten Werkzeuge zur Benutzerverwaltung weiter eingesetzt werden können.

Nicht verschwiegen werden soll hier, dass sich daraus, dass Samba Windows NT (und nicht Windows 2000) nachbildet, auch einige Einschränkungen hinsichtlich der

Administration von Windows-basierten Clients ergeben können: Microsoft hat mit Active Directory die sogenannten „Group Policy Objects“ eingeführt. Damit lassen sich zentral unterschiedliche Teile von Client-Computer- und Benutzerkonfiguration realisieren. Mit einer Windows NT- (oder einer Samba-)Domäne steht dieses Verfahren nicht zur Verfügung. Allerdings lassen sich ähnliche Verfahren die zum gleichen Ziel führen auch mit dem älteren Netzwerkbetriebssystem und in vielerlei Hinsicht einfacher und flexibler mit Samba realisieren. Beispielsweise können Samba-seitig beliebige Skripts ausgeführt werden, wenn sich ein Benutzer an einem Windows-basierten Client anmeldet. Die Skripts können u.a. den Benutzernamen, die Gruppenmitgliedschaften des Benutzers, das Client-Betriebssystem auswerten und haben selbstverständlich Zugriff auf das Verzeichnis, um von dort beliebig weitere Einstellungen zu beziehen. So lassen sich z.B. höchst flexibel Anmeldeskripte oder Registry-Einstellungen generieren, die dann auf dem Client wirksam werden. Die hier vorhandene Flexibilität gleicht den Möglichkeiten von Group Policies und übersteigt diese in einigen Bereichen, sodass der Einsatz von Linux, Samba und OpenLDAP-basierten Domänencontrollern auch hinsichlich der Client-Administration in Umgebungen Sinn macht, in denen zunächst ausschließlich Windows-basierte Clients zum Einsatz kommen sollen.

3.2 Datenbanken

Im Gegensatz zu den File-, Print- und Authentifizierungsdiensten ist die Migration von Datenbanken in der Praxis oft weniger einfach. Mit SQL steht zwar ein Standard-Protokoll zum Zugriff auf Datenbanken zur Verfügung, allerdings verwenden alle Datenbanken unterschiedliche Dialekte und Eigenschaften von SQL. Im Microsoft-Umfeld sind oft zwei Datenbank-Typen anzutreffen: Microsoft-Access Datenbanken und Datenbanken auf der Basis von Microsoft SQL-Server.

Access ist ein besonderer Fall: Zum einen handelt es sich bei dem Programm um ein Frontend für Datenbanken und zum anderen implementiert es selbst eine einfache Datenbank, die jedoch mit einer Reihe von Schwächen versehen ist. Dort wo Access selbst als Datenbank eingesetzt wird, sind keine speziellen Serverdienste möglich. Access legt Datenbanken in Form von Dateien auf der lokalen Festplatte des Clients oder auf einem Fileserver ab. Im Fall eines Fileservers ist es unerheblich ob dafür ein Windows-Betriebssystem oder Samba zum Einsatz kommt. So lange auf dem Client weiter Windows eingesetzt wird, ist es also durchaus möglich, Access in dieser Form weiter zu verwenden. Auf Grund der konzeptionellen Mängel solcher Datenbanken ist jedoch zu prüfen, ob an Stelle dessen nicht auf eine „echte“ Datenbank gewechselt werden soll. Insbesondere für die freie Datenbank PostgreSQL stehen einfach gangbare Methoden zur Migration von Access-Datenbanken zur Verfügung. Nach der Migration der Datenbanken ist die Weiterverwendung von Access als Frontend zur Datenbank möglich.

Werden andere proprietäre, Windows-basierte Datenbanken (z.B. MS SQL-Server oder Oracle) verwendet, muss die betreffende Datenbank selbst nach Linux migriert werden. Besonders einfach ist das, wenn es sich wie bei Oracle um ein Produkt handelt, das für beide Plattformen verfügbar ist. Bei MS SQL-Server ist mit ei-

ner Verfügbarkeit für Linux zur Zeit nicht zu rechnen. Außerdem wird bei solchen Migrationen OSS nicht auf der Ebene der Datenbank eingeführt.

Allerdings stehen für Linux auch freie Datenbanken zur Verfügung, die es mit den proprietären Lösungen unter Windows durchaus aufnehmen und diese in vielen Bereichen schlagen können. Neben den gut bekannten, freien Datenbanken MySQL und PostgreSQL ist hier die ebenfalls freie Datenbank SAP DB zu nennen, auf die auch komplexe MS-SQL-basierte Datenbanken migriert werden können.

Eine generelle Aussage, wie proprietäre Datenbanken auf freie Datenbank-Managementsysteme migriert werden können, kann nicht gegeben werden, weil sich die Anforderungen an eine Datenbank, die benutzten Verfahren zur Kommunikation mit der Datenbank und die verwendeten Features des vorhandenen Systems in der Praxis stark unterscheiden können.

3.3 Groupware

Unter einer Groupware-Lösung wird eine Software verstanden, die es ihren Benutzern mindestens ermöglicht, sich gegenseitig Nachrichten zu senden sowie gemeinsame Kalender und Kontaktlisten zu pflegen. Im Umfeld von Windows-NT-basierten Infrastrukturen gehört das Microsoft-Produkt Exchange 5.5 zu den sehr verbreiteten Groupware-Lösungen. Neben der genannten Basisfunktionalität unterstützt das Produkt außerdem öffentliche Ordner und eine Reihe weiterer Features. Während Microsoft Exchange 5.5 einen eigenen, eingebauten Verzeichnisdienst verwendet, setzen die Nachfolger-Produkte (Exchange 2000/2003) zwingend das Vorhandensein eines Active Directory voraus.

Microsoft Exchange lässt sich unter Linux mittlerweile durch unterschiedliche – freie und proprietäre – Groupwarelösungen ersetzen. Die meisten dieser Lösungen ermöglichen es auch, den Exchange Client, das Programm Microsoft Outlook, weiter zu verwenden und mit diesem auf Mail, Kalender und Kontakte sowie weitere Eigenschaften der Lösung zuzugreifen. Daneben ist es möglich, von Linux-basierten Clients auf dieselben Funktionen zuzugreifen. Dadurch wird die Möglichkeit zur Zusammenarbeit während einer möglichen Client-Migrationsphase erheblich vereinfacht.

Exemplarisch sollen hier zwei für Linux verfügbare Groupware-Lösungen genannt werden. Zum einen die freie Software Kolab und zum anderen das proprietäre Produkt Samsung Contact:

Kolab ist aus einem vom Bundesamt für Sicherheit in der Informationstechnologie finanzierten Projekt hervorgegangen, dessen Ziel darin bestand, auf der Basis vorhandener Komponenten eine Groupwarelösung zu schaffen, auf die komfortabel und mit vollem Funktionsumfang sowohl von Linux- als auch von Windows-basierten Clients aus zugegriffen werden kann. Das Produkt selbst ist freie Software, für die Verwendung mit Microsoft Outlook wird allerdings ein Plugin für Outlook benötigt, bei dem es sich um kostenpflichtige, proprietäre Software handelt. Für Linux ist ein freier Client verfügbar. Kolab verwendet als zentrale Datenbank ein LDAP-Verzeichnis, sodass die Integration in einen „Single-Point-of-Administration“ möglich ist. Das Produkt ist relativ neu, weswegen Erfahrungen über den erfolgreichen Einsatz bisher nur im begrenzten Umfang vorliegen. Aller-

dings lassen die Komponenten, auf denen das Produkt beruht und die sich allesamt in der Praxis schon in sehr großen Installationen bewiesen haben, keine prinzipiellen Probleme vermuten.

Das zweite Produkt, Samsungs Contact, ist per Aufkauf aus dem HP-Produkt OpenMail hervorgegangen. Samsung Contact ist ein ausgereiftes und Praxis-erprobtes, aber proprietäres und kostenpflichtiges Produkt, das bedenkenlos auch in sehr großen Installationen eingesetzt werden kann.

3.4 Terminalservices

Ein Bereich, der auf den ersten Blick nicht der Migration von Windows nach Linux zuzuordnen ist, ist die Einführung von Windows-Terminalservices. Windows-Terminalservices sind eine Komponente der Serverversionen von Windows 2000/2003, die es erlaubt, dass mehrere Benutzer gleichzeitig an einem Windows-basierten Server angemeldet sind und dort interaktiv, graphische Programme ausführen. Dabei verwenden die Benutzer ein Client-Programm, welches Maus- und Tastaturaktionen an den Terminalserer übermittelt und die vom Server zurückgemeldeten Bildschirminhalte auf dem lokalen Rechner darstellt. Für den Benutzer hat es also den Anschein, als würde er direkt mit Programmen auf seinem eigenen Rechner arbeiten.

Terminalservices können bei der Konsolidierung von IT-Infrastrukturen eine entscheidende Rolle spielen, weil durch sie die Installation, Pflege und Aktualisierung von Software auf den Clients unnötig wird. Anwendungen müssen nur zentral auf den Terminalservern installiert werden und können dann sofort von allen Clients aus benutzt werden. Außerdem arbeiten Benutzer über die Terminalservices nicht mehr auf ihrem eigenen PC, bei entsprechender Konfiguration wird es deswegen sehr viel schwieriger, Daten in Bereichen abzulegen, wo sie nicht gesichert werden (wie der lokalen Festplatte). Durch die Einführung von Terminalservices werden also zunächst einmal die administrativen Kosten gesenkt und die Datensicherheit erhöht. Darüber hinaus bieten Terminalservices den strategischen Vorteil, dass sie auch von Linux-basierten Clients aus verwendet werden können.

Ebenso ist es natürlich auch möglich, Linux-basierte Terminalserver einzusetzen, um graphische Anwendungen, die dort ausführbar sind, schnell und einfach und ohne hohen administrativen Aufwand einer großen Zahl von Benutzern bereitzustellen. Linux beherrscht das Terminalserver-Prinzip, seitdem es dort überhaupt eine graphische Benutzeroberfläche gibt. Zur Kommunikation zwischen eigentlichem Anwendungsprogramm und der Software, die für die Ansteuerung von Bildschirm, Tastatur und Maus zuständig ist, wird dort immer ein Netzwerkprotokoll verwendet, sodass Anwendung und Software zur Bildschirmsteuerung nicht auf dem gleichen Rechner ausgeführt werden müssen. Zugriffssoftware auf Linux-Terminalservices steht für alle gängigen Betriebssysteme zur Verfügung.

4 Migration der Clients

Die Migration der Clients ist oft ein deutlich emotionaler diskutiertes Thema als die Migration der Server. Warum? Der Client und der auf ihm ausgeführte Desktop

sind die für alles Benutzer sichtbaren Schnittstellen einer IT-Infrastruktur. Hier sind – anders als bei den Servern, die man Fachleuten überlässt – die Ängste weitaus größer, mit einem neuen oder anderen System nicht zurechtzukommen und das mühsam erworbene Wissen nicht mehr einsetzen zu können.

Allerdings stehen für Linux mittlerweile schon seit einiger Zeit komfortable Desktop-Umgebungen zur Verfügung, die sich weitgehend an den von Windows bekannten Schnittstellen orientieren und sich so verhalten, wie Benutzer es von ihnen erwarten.

4.1 Thin oder fat clients

Eine Umstellung der Clients auf ein neues Betriebssystem gibt die Chance, grundsätzlich über die Betriebsart der Clients nachzudenken. Prinzipiell können zwei Typen von Clients unterschieden werden: *fat clients* und *thin clients*.

Bei *fat clients* handelt es sich normalerweise um klassische Desktop-PCs. Charakteristisch für *fat clients* ist, dass sie mit einer eigenen Festplatte ausgestattet sind, auf denen sich das Betriebssystem, die von den Benutzern zu verwendenden Programme sowie in vielen Fällen auch Benutzerdaten befinden. *Fat clients* zeichnen sich durch hohen administrativen Aufwand (die Software muss auf den Clients installiert und dort aktuell gehalten werden), relativ hohe Hardwarekosten und eine Reihe von potentiellen Gefahren aus. Dazu gehört das Speichern von Daten auf Client-Festplatten, die nicht gesichert werden oder das Verändern der Client-Konfiguration durch Benutzer. Natürlich haben *fat clients* auch Vorteile: Die Rechenleistung dieser Geräte steht allein dem Benutzer zur Verfügung, der gerade damit arbeitet. Außerdem kann es den Benutzern von *fat clients* in einem gewissen Umfang erlaubt werden, diese Geräte selbst zu administrieren, weil sie bei Fehlern in den meisten Fällen nur selbst davon betroffen sind. Ob die Administration oder Konfiguration von Clients gewünscht und sinnvoll ist, ist eine andere Frage, die sicherlich in Abhängigkeit von den Aufgaben des betreffenden Mitarbeiters zu klären ist.

Thin clients haben keine eigene Festplatte. Deswegen können sich auf ihnen weder Benutzerdaten noch persönliche Konfigurationen befinden. So kann einfacher gewährleistet werden, dass Daten dort gespeichert werden, wo sie auch gesichert werden. Außerdem werden Benutzer durch den Einsatz von thin clients unabhängig von „ihrem" PC. Unabhängig davon, wo sie sich an das System anmelden, bekommen sie immer ihre vertraute Umgebung mit allen Programmen und Daten. *Thin clients* sind dafür noch viel mehr als *fat clients* von der korrekten Funktionsweise und Erreichbarkeit der Server abhängig. Falls es nur einen Server gibt, auf dem sich die von den *thin clients* aus verwendeten Anwendungen befinden und dieser Server nicht mehr verfügbar ist, dann können die entsprechenden Anwendungen von keinem *thin client* mehr genutzt werden. Serversysteme auf denen sich Anwendungen für *thin clients* befinden oder auf denen Terminalservices ausgeführt werden, sind also ausreichend verfügbar und redundant auszulegen.

Ob die Entscheidung nun für *fat* oder *thin clients* ausfällt, ist abhängig von der konkreten Situation einer Organisation. Werden viele ähnlich ausgestattete Arbeitsplätze verwendet, an denen die Benutzer hauptsächlich mit Anwendungen arbeiten, die normalerweise nur auf Aktionen der Benutzer warten (wie z.B. Textverarbeitung

oder Datenbankfronten) ist der Einsatz von *thin clients* uneingeschränkt zu empfehlen. In einem Umfeld sehr heterogener Clients in dem es für die Benutzer notwendig ist, regelmäßig Systemkonfigurationen zu verändern oder Programme zu installieren, macht der Einsatz von *thin clients* weniger Sinn, weil Fehler die Gesamtverfügbarkeit des Systems gefährden können.

4.2 Standardanwendungen (Office-Paket)

Wie bereits angesprochen, stehen für Linux verschiedene Benutzeroberflächen in hoher Qualität zur Verfügung. Bei richtiger Einführung einer optimal an die Bedürfnisse der unterschiedlichen Benutzergruppen einer Organisation angepassten Linux-basierten Oberfläche kann hier mit hoher Akzeptanz, höherer Produktivität der Mitarbeiter und vollständiger Funktionalität gerechnet werden.

Etwas komplexer ist die Situation bei der Ablösung des Microsoft Office-Pakets durch ein Linux-basiertes Office-Programm. Für Linux stehen mehrere Office-Pakete zur Verfügung, die unterschiedlich ausgereift sind und einen differierenden Funktionsumfang aufweisen. Als Standard unter den Linux-Offices hat sich mittlerweile *OpenOffice.Org* etabliert. OpenOffice.Org ist die freie Variante des von Sun vertriebenen proprietären Pakets *StarOffice*. Beide Programme sind im Wesentlichen identisch und vollständig kompatibel zueinander, allerdings sind in StarOffice ein paar Komponenten, wie proprietäre Rechtschreibprüfung und eine Datenbank enthalten, die bei OpenOffice.Org fehlen. Die Funktionalität solcher Komponenten ist in OpenOffice.Org allerdings durch andere freie Programme ergänzt worden, sodass sich für den Anwender hinsichtlich des Funktionsumfangs nur wenige Unterschiede ergeben.

OpenOffice.Org ist ein vollständiges, zu Microsoft Office kompatibles Paket bestehend aus Textverarbeitung, Tabellenkalkulation, Präsentations- und Zeichenprogramm sowie einigen kleineren Komponenten wie Formel-Editor oder Diagramm-Werkzeug. Die Kompatibilität erstreckt sich auf zwei wichtige Bereiche. Zum einen werden die Dateiformate von Microsoft Office unterstützt, sodass sich vorhandene Dokumente oder solche von Partnern oder Kunden mit dem Programm öffnen und bearbeiten lassen. Auch das Speichern von Dokumenten in den Microsoft-Formaten wird unterstützt. Dadurch ist es möglich, weiterhin problemlos mit Partnern zu kommunizieren, die das Microsoft-Office-Paket einsetzen. Zum anderen orientieren sich Menüführung, Erscheinungsbild und Bedienkonzept an den Microsoft-Office-Programmen. Anwendern ist es deswegen in der Regel mit äußerst geringem Einarbeitungsaufwand möglich, mit OpenOffice.Org sehr schnell wieder die gleiche Produktivität zu erreichen, wie vor der Umstellung.

Hinsichtlich der Kompatiblität der Dateiformate müssen allerdings auch einige Einschränkungen erwähnt werden. Größte Schwachstelle des Imports ist die Tatsache, dass in Microsoft-Office-Dokumenten enthaltener Visual Basic Code durch den Import nicht in die in OpenOffice.Org eingebaute Skriptsprache *StarBasic* konvertiert wird. Er bleibt zwar im Dokument erhalten, sodass er während der Bearbeitung eines Word-Dokuments durch OpenOffice.Org nicht verloren geht, ist allerdings in dem Programm nicht nutzbar.

Vor einer Migration nach OpenOffice.Org muss also genau geprüft werden, ob und in welchem Umfang von Visual Basic in Microsoft-Office-Dokumenten Gebrauch gemacht worden ist. In vielen Fällen ist Visual Basic in der Vergangenheit verwendet worden um ganze Arbeitsabläufe in Office-Dokumente zu programmieren. Bei einer Migration nach OpenOffice.Org müssen solche „Anwendungen" neu programmiert werden. Dies ist u.U. mit hohem Aufwand verbunden, birgt für die betroffene Organisation allerdings die Chance, solche Anwendungen auf eine sinnvollere und vom Office-Paket getrennte Plattform zu bringen.

Ein entscheidender Vorteil von OpenOffice.Org ist die Tatsache, dass es auch für Windows erhältlich ist. Unternehmen und Behörden können das Programm deswegen Mitarbeitern kostenlos zur Verfügung stellen und diese damit in die Lage versetzen, zuhause mit der selben Software wie am Arbeitsplatz zu arbeiten, ohne sich mit einem Wechsel des Betriebssystems auf ihrem Heim-PC beschäftigen zu müssen. Genauso ist die Bereitstellung für Kunden und Partner möglich, sodass Organisation mit OpenOffice.Org erstmals die Möglichkeit erhalten, ohne Lizenzkosten eine freie Plattform zum Austausch und zur Bearbeitung von Dokumenten aufzubauen.

4.3 Kommunikation und Internet

Die wichtigsten für Linux erhältlichen Internetbrowser sind Mozilla und Konqueror. Mozilla ist die Weiterentwicklung des gut bekannten Browsers Netscape, der u.a. auch für Windows- oder Apple-Systeme verfügbar ist. Neben dem eigentlichen Webbrowser enthält Mozilla u.a. ein E-Mail-Programm, einen Newsreader ein Adressbuch sowie ein Programm zur Erstellung und Bearbeitung von Webseiten.

Der Browser Konqueror ist Teil der Desktop-Umgebung KDE. Das Programm passt sich deswegen gut in KDE-basierte Arbeitsoberflächen ein und benötigt relativ wenig zusätzliche Ressourcen, wenn KDE ohnehin geladen ist. Allerdings gilt Mozilla zur Zeit als der vollständigere Browser.

Sowohl Mozilla als auch Konqueror können die große Anzahl der für Linux erhältlichen Browser-Plugins verwenden. Zu dieser Liste gehören Plugins zum Anzeigen von PDF-Dateien, Java-Plugins, das Realplayer-Plugin oder *Shockwave Flash*.

Als E-Mail und Groupware Clients sollen hier zwei Anwendungen genannt werden. *Ximian Evolution* ist ein in Bedienung und Funktionsumfang weitgehend zu Microsoft Outlook kompatibles Programm, das von Anwendern ohne großen Umstellungsaufwand sofort eingesetzt werden kann. Als alleinstehendes Programm bietet Evolution einen sehr komfortablen E-Mail-Client, eine Komponente zum Arbeiten mit privaten und öffentlichen Adressbüchern, Aufgabenverwaltung und eine Komponente zur Verwaltung von Kalendern. Einladungen für gemeinsame Termine werden (MS-Outlook-kompatibel) per E-Mail versandt. Eine proprietäre Erweiterung zu Evolution erlaubt es außerdem, mit dem Programm, einen MS Exchange Server als Groupware-Server zu verwenden.

Relativ neu ist der KDE-basierte Kolab-Client (ursprünglich bekannt unter dem Namen Kroupware), dessen Einsatz vor allem dann Sinn macht, wenn serverseitig die freie Groupware-Lösung Kolab verwendet wird. Kroupware unterstützt u.a. E-Mail, private und öffentliche Kontaktlisten, sowie Notizen, Aufgabenlisten und eine

der Exchange/Outlook-Kombination vergleichbare Kalenderfunktionalität. Weil auf den Kolab-Server mit einem Exchange-Plugin auch von Microsoft Outlook zugegriffen werden kann, ist damit auch die problemlose Zusammenarbeit von Benutzern mit Linux- und Windows-basierten Arbeitsplatzsystemen möglich.

4.4 Integration von Legacy-Anwendungen

In den meisten Organisationen gibt es so genannte Legacy-Anwendungen, zumeist ältere Programme, die nicht ohne weiteres unter Linux ausgeführt werden können. Dazu gehören meist Windows-basierte Anwendungen, oft aber auch Host- oder Mainframe-basierte Applikationen. Zum Zugriff auf Hosts und Mainframes stehen unter Linux eine Reihe freier und proprietärer Anwendungen bereit, sodass die Integration in der Regel wenig Probleme bereitet. Ob ein freies Programm eingesetzt werden kann oder ein proprietäres Programm eingesetzt werden muss, ist abhängig von den auf dem Host ausgeführten Programmen und muss im Einzelfall analysiert werden.

Ein wenig aufwändiger ist die Integration Windows-basierter Anwendungen. Allerdings hat sich auch in diesem Bereich in letzter Zeit viel getan, sodass es dabei heute keine prinzipiellen Schwierigkeiten mehr gibt. Die Möglichkeiten zur Integration von Windows-Anwendungen lassen sich grob in drei Gruppen einteilen:

1. Integration über Windows-basierte Applikationsserver. In Betracht kommen die Terminal-Services der Server-Versionen von Windows 2000/2003 und das Produkt Citrix, dass einige Erweiterungen und qualitative Verbesserungen zu den Terminalservices beinhaltet. Der Einsatz von Citrix ist allerdings mit nicht unerheblichen zusätzlichen Kosten verbunden und in vielen Situationen nicht wirklich notwendig.
2. Integration über PC-Emulationssoftware. Bei diesem Ansatz kommt Software zum Einsatz, die unter Linux weitere, virtuelle PCs bereitstellt, in die sich andere PC-kompatible Betriebssysteme installieren lassen. Wichtigstes Produkt in diesem Bereich ist die proprietäre Software *VMware*, mit der unter Linux alle Windows-Versionen ausgeführt werden können. Dadurch wird eine extrem hohe Kompatiblität gewährleistet und es lassen sich auch solche Anwendungen bereitstellen, die auch mit neueren Versionen von Microsoft Windows Probleme bereiten. Das allerdings zu einem hohen Preis: Neben der Software zur Bereitstellung der virtuellen Maschinen werden zusätzlich Lizenzen für die Microsoft-Betriebssysteme benötigt. Jedoch zwingen die Kompatiblitätsprobleme der Microsoft-Betriebssysteme auch unter Windows gelegentlich dazu, diesen Weg zu gehen. Abhilfe schafft nur die Anpassung oder Neu-Implementierung der betreffenden Legacy-Anwendungen.
3. Nachbildung des Windows-Betriebssystem unter Linux. Diese Aufgabe hört sich zunächst gewaltig an und das ist sie bestimmt auch. Allerdings ist es dem WINE-Projekt gelungen, mit der gleichnamigen freien Software unter Linux eine Umgebung bereitzustellen, in der sich sehr viele Windows-Anwendungen ausführen lassen, ohne dass Lizenzen für Microsoft-Windows benötigt werden. Zu diesen Anwendungen zählen unter anderem die Office-

Pakete von Microsoft, der Lotus Notes Client, Adobe Photoshop und eine ganze Reihe Windows-basierter Computerspiele, um nur einige zu nennen.

Wie so oft, kann auch hier keine allgemeingültige Empfehlung gegeben werden, welche Variante für eine Organisation am sinnvollsten geeignet ist. Grundsätzlich ist die Verwendung von WINE allerdings am wirtschaftlichsten, weil keine zusätzlichen Server oder Betriebssystemlizenzen benötigt werden und die Integration von Windows-Anwendungen in den Linux-Desktop mit WINE am besten möglich ist. In Migrationsprojekten lohnt es sich also, die Kompatiblität von Windows-basierten Legacy-Anwendungen mit WINE eingehend zu testen.

4.5 Strategie bei der Desktop-Migration

Die von jedem Mitarbeiter einer Organisation einzusetzenden Werkzeuge und insbesondere „der Rechner" sind ein Thema, bei dem die Einbeziehung der Betroffenen unvermeidlich ist. Organisationen unterscheiden sich zwar sehr stark hinsichtlich des notwendigen Umfangs der Einbeziehung von Mitarbeitern, allerdings sind Desktop-Migrationen, die ohne die Mitwirkung von Benutzern durchgeführt werden, mit hohen Risiken behaftet.

Bei der Migration des Desktops ist es sinnvoll, die Betroffenen früh zu informieren, ständig auf dem Laufenden zu halten und auf Sorgen und Wünsche einzugehen. In vielen Unternehmen und Behörden ist es sogar zwingend notwendig, den Betriebsrat bei wichtigen Entscheidungen mit einzubeziehen.

Es kann an dieser Stelle nur dazu aufgefordert werden, die Desktop-Anwender als Experten für ihren Arbeitsplatz anzusehen. Die Benutzer wissen am besten, was ihnen an ihrem Arbeitsgerät gut oder schlecht gefällt, wo sie dringenden Verbesserungsbedarf sehen und welche Eigenschaften unbedingt beibehalten werden müssen.

Ein sinnvolles Vorgehen bei der Einführung von Linux-basierten Desktop-Systemen besteht in folgenden Schritten:

1. Öffentliche Bekanntgabe des Vorhabens, den Einsatz von Linux als Desktop-Betriebssystem zu prüfen.
2. Einladung zu einer Informationsveranstaltung, in der die Gründe für diese Prüfung dargestellt werden und der Linux-Desktop vorgestellt wird. Dies sollte in Form einer Präsentation geschehen. Zusätzlich ist es sinnvoll, an einem für alle Mitarbeiter zugänglichen Ort, bereits zu diesem Zeitpunkt Linux-basierte PCs aufzustellen, damit Interessierte die Möglichkeit haben, das neue System direkt auszuprobieren. Wichtig ist es, Offenheit für Verbesserungen zu signalisieren. Linux ist als freies Betriebssystem sehr viel leichter anpassbar als proprietäre Systeme dies je sein können. Diesen Umstand kann man sich zu Nutze machen und sinnvolle Vorschläge direkt in die endgültig implementierte Lösung einfliessen lassen.
3. Nach Abschluss der Informationsveranstaltung wird auf freiwilliger Basis ein Expertenteam „Desktop" zusammengestellt, welches die Implementierer bei der Gestaltung der neuen Lösung unterstützt. Diese Experten werden zu

wichtigen Bereichen befragt und gebeten, die ständig verbesserten Prototypen zu testen und Verbesserungsvorschläge zu machen.

4. Das Migrationsteam implementiert dann auf der Basis der Konzeption einen ersten Prototypen und stellt diesen der Expertengruppe zur Verfügung. Daraufhin werden Verbesserungsvorschläge geprüft und ggf. in die Lösung eingearbeitet. Dieser Prozess ist iterativ und muss u.U. mehrfach wiederholt werden.
5. Nach Abschluss der Entwicklung des prototypischen Desktops und Entscheidung über die Machbarkeit erfolgt die öffentliche Bekanntgabe des Vorhabens, Linux nun tatsächlich als Desktop-Betriebssystem einführen zu wollen. Darin wird das weitere Vorgehen, nämlich Test und Optimierung mit einer Pilotgruppe und anschließendes Roll-Out beschrieben.
6. Schließlich wird der neue Desktop bei einer Gruppe von Pilot-Benutzern installiert. Welche Gruppe dafür in Frage kommt, ist abhängig von der betreffenden Organisation. Allerdings ist es sinnvoll eine Gruppe von Mitarbeitern zu wählen, die auch sonst eng miteinander zusammen arbeiten, damit es kurze Kommunikationswege und einfache Möglichkeiten, sich bei Problemen gegenseitig zu helfen, gibt. Hilfreich ist es auch, wenn sich in der Gruppe der Pilot-Anwender Mitarbeiter aus der Expertengruppe befinden. Diese haben bereits Erfahrungen mit dem neuen System gesammelt, die im Piloten sinnvoll verwendet werden können. Auch die Pilotgruppe sollte auf Basis von Freiwilligkeit ausgewählt werden. Ein Vorgehen besteht darin, ein Bewerbungsverfahren zu verwenden, an dem Abteilungen teilnehmen können. Viele IT-Abteilungen von Organisationen neigen dazu, sich selbst als Pilotgruppe für neue Software anzubieten. In vielen Bereichen ist das auch sinnvoll, bei der Einführung eines neuen Desktops-Systems sollte auf die Einbeziehung „normaler" Benutzer allerdings nicht verzichtet werden.
7. Die mehrwöchige Pilotphase muss eng begleitet werden. Dazu gehören die Einrichtung eines zumindest während der Kernzeiten verfügbaren Help-Desks, der bei Aufteten etwaiger Probleme schnell und kompetent helfen kann und die regelmäßige Befragung der Pilot-Benutzer über ihre Erfahrungen mit der Benutzung des neuen Systems. Befragungsergebnisse und Aktivitäten des Help-Desks werden ausgewertet und notwendige Verbesserungen werden in die Lösung integriert. Solche Änderungen müssen dann wieder durch die Pilotgruppe getestet werden.
8. Vor dem Roll-Out kann eine weitere Informationsveranstaltung durchgeführt werden, in welcher der neue Desktop öffentlich präsentiert wird. In dieser Veranstaltung sollte ausreichend Zeit für Fragen und Antworten zur Verfügung stehen. Besonders günstig ist es, wenn Fragen von den Mitgliedern der Experten- und Pilotgruppen beantwortet werden können.
9. Selbstverständlich muss auch der eigentliche Roll-Out gut vorbereitet werden. Auch hier bietet sich ein abteilungsweises Vorgehen an. Zunächst können in einer zu migrierenden Abteilung je nach Größe ein oder zwei Testarbeitsplätze aufgestellt werden, die von den Mitarbeitern genutzt werden können, um die Arbeit mit dem neuen System kennenzulernen. Falls notwendig,

erfolgt während dieser Phase auch die Schulung der Abteilungsmitglieder. Danach werden die Arbeitsplatzrechner umgestellt. In dieser Phase ist ebenfalls mit häufiger Beanspruchung des Help-Desks zu rechnen, weswegen dieser personell entsprechend gut ausgestattet sein muss.

10. Auch nach dem Roll-Out werden sich weitere Möglichkeiten zur Verbesserung zeigen. Deswegen ist es sinnvoll, das Projekt weiter zu begleiten und nach Ablauf einer gewissen Zeit Updates und Optimierungen vorzunehmen, die wieder zunächst bei einer Pilotgruppe getestet werden sollten, bevor sie organisationsweit installiert werden.

Das beschriebene Verfahren ist aufwändig und sicherlich erst ab einer bestimmten Organisationsgröße in dieser Form zu rechtfertigen. Es kann deswegen nicht als strikter Vorgehensplan für alle Unternehmen oder Behörden dienen, sondern muss immer an die spezielle Gegebenheiten einer Organisation angepasst werden. Wichtig ist aber die Feststellung, das es für den Erfolg von Migrationsprojekten im Desktop-Bereich unerlässlich ist, die Betroffenen mit einzubeziehen und ihnen die Chance zur Mitgestaltung ihres Werkzeuges zu geben.

5 Fazit

Linux ist reif für den Enterprise-Einsatz. In vielen Bereichen von Serveranwendungen schon seit sehr langer Zeit und nun auch auf dem Desktop. Unternehmen und Behörden können durch den Einsatz von Linux und anderer OSS die Sicherheit, die Stabilität, die Flexibilität, vor allem aber die Wirtschaftlichkeit ihrer IT-Infrastrukturen erheblich steigern. Gleichzeitig befreien sie sich durch den zunehmenden Einsatz freier Software aus der Abhängigkeit von Softwareherstellern, die in vielen Fällen teuer bezahlt werden müssen.

In welchem Umfang und in welchen Bereichen der Einsatz von Linux für Organisationen Sinn macht, kann nur eine detaillierte Analyse der Ist-Situation und der strategischen Ziele der betreffenden Organisation ergeben. Insofern ist eine solch eingehende Analyse unerlässlich bei der erfolgreichen Einführung auf Open Source basierender Software.

Die Migration im Serverraum wird mehr und mehr zu einer Frage nach der richtigen technischen Vorgehensweise. Prinzipielle Probleme sind hier nur in besonderen Fällen zu erwarten. Allerdings ist immer auch dem menschlichen Faktor gebührende Beachtung zu schenken. Administratoren, Systembetreuer und auch Help-Desk-Mitarbeiter müssen ihrem Arbeitsmittel positiv gegenüber stehen und gut damit arbeiten können. Deswegen sind Schulungen, Einbeziehung der Administratoren bei Entscheidungen und gelegentlich auch Betriebsbegleitungen unbedingt erforderlich, um Migrationsprojekte erfolgreich und wirtschaftlich durchzuführen.

Noch viel wichtiger ist die Einbeziehung der Betroffenen bei Desktop-Migrationen, auch weil normale Computerbenutzer sich in einer Organisation für andere Dinge als für optimale IT-Infrastrukturen interessieren (sollen). Deswegen ist hier ein entsprechender Aufwand zu betreiben. Es lohnt sich, denn im Desktop-Bereich

sind die Möglichkeiten, Einsparungen vorzunehmen noch größer als auf der Serverseite.

Literatur

BMI (2003): *Migrationsleitfaden – Leitfaden für die Migration der Basissoftwarekomponenten auf Server- und Arbeitsplatzsystemen*, Bonn: Mitp
online http://www.bmi.bund.de/dokumente/Bestellservice/ix_92583.htm

Univention GmbH (2002): *Migration einer Windows NT-basierten IT-Infrastruktur nach Linux beim Bundeskartellamt,*
online http://www.univention.de/

Kapitel 4

Recht und Politik

Einleitung

MARTIN UNGER

In den letzten Jahren hat sich viel an Rechten und Gesetzen im Bereich von Software und Internet geändert, national wie auch international. Häufig geschah das außerhalb des Wahrnehmungsbereichs der Nutzer, Anwender, Verbraucher, oder wie immer man die Personen vor dem Computer lieber bezeichnen möchte. Die meisten von ihnen haben überhaupt noch nicht oder höchstens am Rande bemerkt, dass Ende des letzten Jahres ein neues Urheberrecht in Kraft getreten ist. Und auf deutscher wie auch auf europäischer Ebene sind weitere Gesetze in Vorbereitung, z. B. auf dem Gebiet der Patentierung der so genannten „computer-implementierbaren Erfindungen“[1] oder zur Durchsetzung von Rechten des „geistigen Eigentums“.[2] Viele dieser Gesetze gelten im Internet und stellen die Open-Source-Community vor neue Herausforderungen.

Open Source ist in der Politik angekommen, kein Zweifel. Zunehmend werden die Potentiale von Freier bzw. Open-Source-Software erkannt. Neben Unternehmen versuchen auch immer mehr staatliche Institutionen, die sich mit Open Source bietenden Chancen zu nutzen. Und der Möglichkeiten sind viele:

- Kosteneinsparungen bei kommunalen Beschaffungen;
- Struktur- und Industriepolitik mit dem Ziel, lokale Produzenten zu unterstützen;
- Monopolstrukturen im Softwaremarkt entgegenwirken;
- Druck auf die Hersteller, sicherere Produkte zu liefern, usw.

In diesem Kapitel wird versucht, den Nutzern von Software einen Einblick in die momentanen und zukünftigen Entwicklungen auf dem Gebiet der Politik und des Rechts mit Bedeutung für Open-Source-Software[3] zu vermitteln, ein Ausblick

---

1 Der Begriff wird in der EU-Bürokratie benutzt. Es handelt sich um mit Software implementierte Verfahren, zu deren Ausführung Computertechnologie zum Einsatz kommt. Diese betreffen, neben Spezialanwendungen wie Antiblockiersteuerungen, in der Regel auch Software die auf einem handelsüblichen PC ausgeführt werden kann. Zum Stand des Verfahrens zur Erarbeitung der Richtlinie vgl. http://www3.europarl.eu.int unter: Protokoll vom 24/09/2003, gestützt auf Dokument A5-0238/2003 – endgültige Ausgabe.

2 Entwurf zur Richtlinie http://europa.eu.int/comm/internal_market/en/indprop/com02-92de.pdf.

3 Ich werde in diesem Text Open-Source-Software und Freie Software der Einfachheit halber synonym verwenden. Das dies laut Richard Stallman nicht ganz richtig ist, kann man hier nachlesen: http://www.gnu.org.

auf die gesellschaftlichen Möglichkeiten des gezielten Einsatzes dieser Methode inklusive.

Patentrecht

In den USA sind Softwarepatente seit der Entscheidung des Supreme Courts im Fall Diamond v. Diehr (1981[4]) Gang und Gäbe und gelten dort zum Sichern von Umsätzen aus computer-implementierbaren Erfindungen und gegenüber Konkurrenten als unverzichtbar. In der EU ist im Moment eine Richtlinie zur Softwarepatenten in der Endphase Ihrer Entstehung – und alles andere als unumstritten.[5]

In den USA sind schon häufig Probleme mit der Patentierung von Algorithmen aufgetreten, wie im Falle des GIF-Formats, auf dessen LZW-Komprimierungsalgorithmus Compuserve (Unisys) ein Patent hält.[6]

Außerdem existieren ganz grundlegende Fragen die es noch zu klären gilt. Im Gegensatz zu großen Softwarehäusern, dem EPA und großen Firmen anderer Sparten[7], die in dieser Richtlinie mehr Investitionssicherheit für Unternehmen in Europa und Anreiz zu Innovationen sehen –

> „Die Konkurrenz in den USA kann Ideen aus dem Internet abkupfern und anschließend auch noch patentieren lassen!“[8]

– betrachten große Teile der von der neuen Richtlinie Betroffenen diese Begründungen mit Skepsis und vermuten hinter diesen, ihrer Meinung nach vorgeschobenen Begründungen, eher die Expansionsabsichten und das Sichern der Pfründe von Monopolisten. Diese Annahme wird durch wissenschaftliche Studien wie z. B. von Bessen & Hunt[9] gestützt.

In den USA ist die Patentierung von Software und deren Bausteine in großen Firmen generalstabsmäßig organisiert. Hierzu unterhalten Unternehmen wie z.B. Oracle, Microsoft und IBM große Rechtsabteilungen um Neuerungen patentieren zu lassen, evtl. alte patentierbare Verfahren zu finden, Verletzungen ihrer Patente festzustellen und diese zu verfolgen. Hier stellt sich die Frage ob kleinere Firmen sich dies ebenfalls leisten könnten, und ob nicht immer wieder Programmierer in eine „Pantentfalle“ („U-Boot-Patente“) geraten, sollten sie Verfahren bzw. Algorithmen verwenden die, ohne ihr Wissen, patentgeschützt sind.

In den EU-Staaten wird das Patentrecht im Bezug auf Software sehr unterschiedlich ausgelegt, und regelmäßig wurden auf nationaler Ebene Patente auf Software abgelehnt. Nun hat aber das Europäische Patentamt (EPA[10]) in den letzten Jahren Entscheidungen in Sachen Softwarepatente getroffen, die sehr zwiespältige

4 Für einen kurzen Abriß zur Geschichte der Softwarepatente siehe Gehring und Kretschmer (2003).
5 Vgl. http://swpat.ffii.org/papiere/eubsa-swpat0202/index.de.html.
6 Ein Dateiformat für Grafiken; mehr zu diesem Fall: http://lpf.ai.mit.edu/Patents/Gif/Gif.html.
7 Z. B. Amazon.
8 Helmuth Gümbel vom Beratungsunternehmen Strategy Partners.
9 Siehe auch http://www.researchoninnovation.org/swpat.pdf.
10 Das Europäische Patentamt ist keine EU-Institution. Allerdings sind alle EU-Staaten dem Euopäischen Patentübereinkommen (EPÜ) beigetreten und haben so die Kompetenz des EPA zur Rechtsprechung in Patentsachen grundsätzlich anerkannt.

Reaktionen bei den Betroffenen ausgelöst haben. So wurden vom EPA in den letzten Jahren um die 30000 Patente bewilligt,[11] die als computer-implementierbare Erfindungen bezeichnet werden. Die Rechtslage in den einzelnen EU-Staaten wurde unter anderem durch diese Tatsachen immer unübersichtlicher.

Daraufhin sah die EU-Kommission Handlungsbedarf und so wurde im Februar 2002 der erste Entwurf einer EU-Patentrichtlinie[12] vorgestellt, die für Entscheidungen der Patentämter in den einzelnen Staaten der EU eine gemeinsame Rechtsgrundlage schaffen soll.

Diese Richtlinie war und ist aber unter den Betroffenen sehr umstritten. Vorrangig kleinere Softwarehäuser und Produzenten von Open-Source-Software (OSS) sehen den rechtlichen Änderungen mit Befürchtungen entgegen. Bei ihnen hat der Richtlinienvorschlag zum Teil deutliche Kritik und Unverständnis ausgelöst. Sie bezeichnen die vorgeschlagenen Formulierungen als viel zu schwammig und kritisieren, dass zwar einige wichtige Einschränkungen angesprochen würden, aber viel zu ungenau umrissen würde, was denn nun wirklich patentierbar sei, und was nicht.

Folgende Einschränkungen tragen beispielsweise zur Verwirrung bei: Software „als solche" soll nicht patentierbar sein, aber falls eine computer-implementierbare Erfindung eine gewisse *Technizität* aufweisen sollte, würde sie patentierbar sein. Dies ist ein schwieriges Thema, da umstritten ist, inwieweit Software selbst technische Eigenschaften besitzt, oder aber im Zusammenspiel mit der Hardware eines PCs quasi automatisch eine technische Komponente erhält.

Verschiedene Akteure, darunter der Förderverein für eine Freie Informationelle Infrastruktur (FFII[13]), eurolinux[14] und das Institut für Rechtsfragen der Freien und Open-Source-Software (ifrOSS[15]), haben Eingaben bei der EU-Kommission eingereicht. Auf Grund dieser Eingaben und der Kritik von Regierungsstellen verschiedener EU-Länder (z.B. Frankreich) an der Novellierung des Patentrechts wurde die Formulierung der Richtlinie zwar in einigen Punkten verändert und etwas präzisiert, allerdings ist die aktuelle Version den vorher genannten Vereinigungen immer noch zu weitgehend und zu ungenau.

Widerstand gegen die Neuregelung kommt ebenfalls von Universitäten und anderen Forschungseinrichtungen, die befürchten, durch neue Patentgesetze in ihrer Arbeit behindert zu werden.

Es existieren noch viele weitere Kritikpunkte an und Schwierigkeiten mit der neuen EU-Patentrechtsrichtlinie. Aus der Sicht eines Patentrechtsexperten geht der Artikel „Patentschutz und Softwareentwicklung – ein unüberbrückbarer Gegensatz?" von *Andreas Wiebe*, Professor an der Wirtschaftsuniversität Wien (Institut für Bürgerliches Recht, Handels- und Wertpapierrecht, Abteilung für Informationsrecht und Immaterialgüterrecht), auf diese Problematik ein. Wiebe untersucht, inwiefern Open Source von der Richtlinie betroffen wäre und durch welche

11 Vgl. FFII-Homepage http://patinfo.ffii.org/.
12 Siehe auch http://europa.eu.int/prelex/detail_dossier_real.cfm?CL=de&DosId=172020 .
13 Vgl. FFII-Homepage http://swpat.ffii.org/.
14 Vgl. FFII-Homepage http://eurolinux.ffii.org/index.en.html.
15 Näheres unter http://www.ifrOSS.de.

Mittel sich Open-Source-Entwickler gegen die Risiken des Patentrechts schützen können.

Urheberrecht

Die meisten Computernutzer in unserem Land haben offensichtlich noch nicht bemerkt, dass in der EU wie auch in Deutschland der zunehmenden Digitalisierung von Werken[16] und deren Verbreitung auch über das Internet durch ein neues Urheberrecht Rechnung getragen werden soll.

Bei der Verabschiedung der Richtlinie 2001/29/EC, „Zur Vereinheitlichung...", diente der Digital Millenium Copyright Act (DMCA) der USA als Vorbild, welcher 1998 in Kraft getreten ist. Die Rechte der Urheber von digitalen Werken sollen mit der neuen Gesetzesvorlage gestärkt werden, aber auch hier regt sich Widerstand gegen die weitreichende Verschärfung der Schutzinstrumente.

In der Debatte um das neue Urheberrecht wird auch deutlich, wie groß der Unterschied zwischen Copyright und Urheberrecht im Grunde ist,[17] und wie schwer deshalb eine Adaption fällt. Während Vertreter der großen Rechteinhaber z.B. in der Musikindustrie die Neuerungen im Urheberrechtsgesetz begrüßen –

> Das neue Gesetz wäre superwichtig für die Industrie, wir erwarten, dass die Umsatzrückgänge zumindest gebremst werden.[18]

–, kritisieren viele Vereinigungen (u.a. die Initiative „Rettet die Privatkopie"[19]) und kleinere Softwarehäuser in Deutschland das neue Gesetz als zu unausgewogen und im Grunde genommen völlig unnötig, da z.B. für die erfolgreiche Verfolgung von Raubkopierern die geltenden Gesetze völlig hinreichend seien.

Dieses Argument wird indirekt durch die Meinung der Musikindustrie gestützt, dass die neuen Gesetze so oder so nur mit Hilfe von Digital-Rights-Management-Systemen (DRM) durchzusetzen wären,[20] welche ihrerseits sehr umstritten sind.[21]

Gegner des Gesetzes bemängeln, dass dem Nutzer zu viele Rechte entzogen würden. Die Stärkung des Urheberrechts der Autoren und der Positionen der Verwerter ginge auf Kosten der Verbraucher. Manche Stimmen verkünden sogar die Behinderung von Innovation und Informationsfreiheit.[22]

Natürlich gibt es noch eine große Anzahl weiterer Argumente bezüglich der Novellierung des Urheberrechts. Auf den Zusammenhang zwischen Open Source und Urheberrecht geht der Artikel „Urheber- und Lizenzrecht im Bereich von Open-Source-Software" von *Axel Metzger* und *Olaf Koglin* (ifrOSS) genauer ein.

---

16 Z. B. Bücher zu E-Books.

17 Vgl. auch http://www.jere-mias.de/biwi/urheb1.html.

18 Hartmut Spiesecke, Leiter der Presse- und Öffentlichkeitsarbeit des Bundesverbandes der Phonographischen Wirtschaft.

19 Näheres unter http://www.privatkopie.net/.

20 Nachzulesen unter http://www.privatkopie.net/files/guennewig230103.pdf .

21 Eine umfangreiche Darstellung der technischen, politischen, rechtlichen und politischen Aspekte findet sich bei Becker, Buhse, Günnewig und Rump, Hrsg., (2003).

22 Siehe http://www.heise.de/newsticker/data/jk-29.01.02-008.

Lizenzen

Softwareentwickler benutzen das Mittel der Lizenzierung um im Verhältnis mit dem Nutzer ihrer Produkte die Nutzungsrechte zu regeln. Der Verbraucher bemerkt davon meistens die während der Installation von Programmen erscheinenden Fensterchen, in denen ein ellenlanger Text und darunter zwei „Buttons" mit der Beschriftung „Akzeptieren" oder „Ablehnen" zu sehen sind. Was der Softwarenutzer im Normalfall nicht weiß ist, dass er mit dem Anklicken des „Akzeptieren-Buttons", in Sekundenschnelle einen rechtskräftigen Vertrag mit dem Urheber jener Software abschließt. Wird eine der Bedingungen des Vertrages verletzt, dann verfällt die Lizenz im Normalfall, und der Nutzer macht sich unter Umständen eines Urheberrechtsbruchs schuldig.

Von Microsofts End User License Agreement (EULA) oder ihrer „Shared Source Licence"[23] über die General Public Licence (GPL)[24] der Free Software Foundation bis hin zur „FreeBSD-Copyright-Licence"[25] gibt es sehr unterschiedliche Ansätze der Lizenzierung. Einige Lizenzen kommen mit einem „Copyleft"-Effekt daher, wie die GPL, die meisten aber ohne solchen.

Lizenzen wie die „Shared Source Licence" von Microsoft geben zwar dem Namen nach vor, liberale Lizenzen zu sein, wie man sie eher im Umfeld der OSS vermuten würde, unterscheiden sich von diesen aber zum Teil deutlich.[26]

Open-Source-Lizenzen

„Copyleft"[27] bedeutet, dass derjenige der eine Software unter einer dementsprechenden Lizenz erhalten hat, diese auch unter derselben wieder weiterverbreiten muss, falls er sich denn zur Weiterverbreitung entschließt.

Im Gegensatz dazu erlauben es OSS-Lizenzen ohne Copyleft, wie z.B. die FreeBSD-Lizenz, dem Nutzer der entsprechenden „Werke," diese unter beliebiger Lizenz, also auch als proprietäre[28] Software, weiter zu verbreiten.

Gemeinsam ist fast allen Lizenzen aus dem OSS-Bereich das Einräumen des Rechts zur Veränderung, Vervielfältigung und Weiterverbreitung des unter diesen Lizenzen veröffentlichten Werks.

Allerdings muss man sagen, dass den speziellen Vorstellungen der Urheber hinsichtlich der eingeräumten Nutzungsrechte und Ausnahmen praktisch keine Grenzen gesetzt sind. Für jeden noch so kleinen Spezialfall gibt es im Bereich der OSS eine mögliche Lizenzierungsform.[29]

---

23 Vgl. auch http://www.microsoft.com/windows/embedded/ce.net/previous/downloads/source/license.asp.

24 Vgl. http://www.gnu.de/gpl-ger.html.

25 Näheres unter http://www.freebsd.org/copyright/license.html.

26 Vgl. http://www.microsoft.com/windows/embedded/ce.net/previous/downloads/source/license.asp.

27 Informationen dazu unter http://www.gnu.org/copyleft/copyleft.html.

28 Siehe http://coforum.de/index.php4?Propriet%E4re_Software.

29 Man muss allerdings erwähnen, dass viele Lizenzen in ihren Formulierungn recht laienhaft sind und einer Prüfung durch einen Juristen wohl kaum Stand halten würden.

Des weiteren gibt es auch zahlreiche Lizenzen, die nicht direkt für Werke aus dem Bereich der Software bestimmt sind, wie z.B. ifrOSS' „Freie Lizenz für Texte und Textdatenbanken".[30]

Diese versuchen, die gesellschaftliche Grundidee der Free Software Foundation[31] für andere Werke zu verwirklichen. Allerdings sind viele dieser so genannten „Open Content"-Lizenzen wiederum sehr unterschiedlich formuliert, und auch hier gibt es inzwischen Lizenzen für fast jeden erdenklichen Anwendungsbereich, von Musik über Werke der bildenden Künste bis hin zu „Open Book"-Lizenzen, die Bücher in greifbarer Form betreffen. Ein innovativer Ansatz, diese verwirrende Vielfalt praktikabel unter ein Dach zu bekommen stammt vom „Creative Commons" Projekt.[32]

Im Text von *Axel Metzger* und *Olaf Koglin* (vom ifrOSS) wird das „Urheber- und Lizenzrecht im Bereich von Open-Source-Software" behandelt. Die Autoren geben eine kurze Einführung in die Konzepte und stellen dar, inwieweit Software im Allgemeinen und Open-Source-Software im Konreten davon erfaßt wird. Sie diskutieren die Frage der Lizenzierung von Open-Source-Software und die daraus resultierenden Rechte und Pflichten der Anwender.

Politik und Open-Source

Vor einigen Monaten wurde in München entschieden, in den nächsten Jahren OSS für die Verwaltung zu nutzen; ebenfalls hat sich der Bundestag zur Installation von Linux auf seinen Servern entschlossen. Da stellt sich doch die Frage warum nach all den Jahren Microsoft nun dieser Sinneswandel? Wie steht es denn mit dem Argument dass OSS zwar in der Anschaffung billiger sei, dieser Preisvorteil aber durch die Schulung der Nutzer in kürzester Zeit aufgefressen sei, wird hier wieder Geld in ein Millionengrab geschaufelt, wie der Bürger schnell argwöhnt?

In den letzten Jahren wurden auch einige OS-Projekte von der Politik gefördert (z.B. BerliOS[33]). Hat dies rein wirtschaftliche Gründe, oder will man tatsächlich die Entwicklung auf dem Gebiet der OSS vorantreiben? Sieht man konkrete Chancen für die Zukunft? Im Koalitionsvertrag der neuen Regierung wurde sogar festgeschrieben, dass eine Benachteiligung von nicht-proprietärer Software nicht geduldet werden darf. Warum ist OSS plötzlich so *en vogue*?

Uwe Küster (MdB),[34]Vorsitzender der IuK-Kommission[35] des Ältestenrates, hat zu diesen Fragestellungen einen Artikel mit dem Titel „Open-Source-Software – Ein Weg aus der Abhängigkeitsfalle zurück zur unternehmerischen Freiheit" beigesteuert. Er beschreibt darin die Prozesse, an deren Ende die Entscheidung des Bundestages für Open-Source-Software gefällt wurde. In seinem Beitrag begründet Küster, welche Überlegungen angestellt wurden und welche Argumente zur Ent-

30 Nachzulesen unter http://www.ifrOSS.de/ifrOSS_html/ifl.html.
31 Siehe auch http://www.gnu.org/philosophy/free-sw.html.
32 Die Homepage des „Creative Commons"-Projekts findet sich unter http://creativecommons.org/.
33 Informationen zu BerliOS: http://www.berlios.de/.
34 Siehe http://www.bundestag.de/mdb14/bio/K/kuestuw0.html.
35 Kommission des Ältestenrates für den Einsatz neuer Informations- und Kommunikationstechniken und -medien.

scheidung geführt haben – und welche Erwartungen für die Zukunft daran geknüpft werden.

E-Democracy

Stellen Sie sich vor, um einen neuen Ausweis zu beantragen oder einen Zweitwohnsitz anzumelden, müssten Sie keine Nummer mehr ziehen um dann zwei Stunden zu warten bis ihre Nummer aufgerufen wird. Anschließend dürfen Sie zwei Minuten mit einer gestressten Mitarbeiterin der Meldestelle sprechen und ihr Anliegen vorbringen. In Zukunft werden Sie dies alles in fünf Minuten vom heimischen PC aus erledigen können.

Oder so ähnlich, glaubt man den Visionen für E-Government. Die Politik hat erkannt, dass die zunehmende Verbreitung des Internets große Möglichkeiten bietet, den Bürger einerseits stärker an politischen Entscheidungsprozessen teilhaben zu lassen und andererseits ganz banale Verwaltungsaufgaben billiger und praktischer zu gestalten.

Der Bund hat hierzu die Initiative „BundOnline 2005" ins Leben gerufen, die zum Ziel hat, alle internetfähigen Verwaltungsaufgaben des Bundes bis 2005 online unter www.bund.de anzubieten. Dank dieser Initiative ist Deutschland auf der Ebene des Bundes im internationalen Vergleich nach einer Studie von Accenture[36] etwas weiter nach oben gerückt, allerdings muss man konstatieren, dass Staaten wie Kanada und die USA auf diesem Gebiet schon um einiges weiter sind.

Auf kommunaler Ebene sieht es hier in Deutschland deutlich schlechter aus. Im europäischen Vergleich liegen wir zurück und die Entwicklung stagniert. Das Problem ist, dass trotz bekannt knapper Kassen alle Kommunen eine eigene Strategie verfolgen und individuelle Konzepte erarbeiten. Des weiteren wird in Deutschland besonders viel Wert auf die Sicherheit der Verfahren gelegt. Dazu geeignete Technologien zu entwickeln und zu etablieren, dauert seine Zeit. Die Kommunen sollten erkennen, dass die Entwicklung billiger würde und schneller voran gehen würde, wenn sie kooperierten.[37]

Es bleibt nur zu sagen, dass es wünschenswert wäre, wenn der Bürger der Politik durch die neuen Möglichkeiten der Vernetzung wirklich näher käme. So könnten viele Menschen in Deutschland vielleicht ein besseres Gefühl dafür bekommen wie wertvoll ihre Demokratie ist, und vielleicht gewännen sie durch die neue „Nähe" zu den Entscheidungsträgern ein wenig mehr den Eindruck, wirklich etwas bewegen zu können.

In unserem Verständnis wird es E-Democracy ohne Open Source nicht geben. Wo es um elektronisch vermittelte Kommunikationsbeziehungen geht, verlangt die Transparenz der Prozesse nach der Transparenz der Mittel. Da dieses Buch ja auch eine Entscheidungshilfe für die Politik darstellen soll, habe ich zu diesem Thema ei-

---

36 Vgl. http://www.accenture.com/xd/xd.asp?it=enweb&xd=newsroom%5Cepresskit%5Ccrmgov%5Cepresskit_crmgov.xml.

37 Genauer wird dies in einer Studie von Cap Gemini Ernst & Young Deutschland (2004) betrachtet, vgl. http://www.ch.cgey.com/servlet/PB/menu/1265568_l1/index.html.

nen Text von *Steven Clift* [38] ausgewählt, in dem er über die momentane Situation und die Möglichkeiten von E-Democracy spricht.

Literatur

Becker, Eberhard, Willms Buhse, Dirk Günnewig und Niels Rump (2003): *Digital Rights Management: Technological, Economic, Legal and Political Aspects*, Lecture Notes in Computer Science, Vol. 2770, Berlin, Heidelberg, New York, Springer Verlag

Gehring, Robert A. und Kretschmer, Martin (2003): *Software-Patente in Vergangenheit, Gegenwart und Zukunft*, FifF-Kommunikation, 20. Jg., Nr. 4, S. 26–31

[38] Siehe http://www.publicus.net.

Patentschutz und Softwareentwicklung – ein unüberbrückbarer Gegensatz?

ANDREAS WIEBE

I. Einleitung

Fast alle Schutzsysteme des Immaterialgüterrechts lassen sich für den Schutz von Software nutzbar machen. Bereits seit Beginn der Verbreitung der Computertechnik standen das Urheberrecht und das Patentrecht im Vordergrund. Dabei zeichnete sich schnell ein Trend zum Urheberrecht als primärem Schutzrecht ab, der durch die Richtlinie zum Urheberrecht von 1991 und die darauf folgende Harmonisierung in Europa verfestigt wurde.[1] In den neunziger Jahren wurde in den USA neben dem Know-how-Schutz auch verstärkt der Patentschutz für Software genutzt, was vor allem im Bereich geschäftlicher Anwendungen zu einigen sehr breiten Patenterteilungen führte.

Dies hat, vor allem auf der Seite der Open-Source-Bewegung, die Befürchtung geweckt, dass auch in Europa eine breite Patentierung von Software erfolgen könnte. Wegen der gegenüber dem Urheberrecht andersartigen Ausgestaltung hinsichtlich Schutzgegenstand und -umfang ist es nicht nur zweifelhaft, ob sich der Patentschutz in gleicher Weise für das Open-Source-Modell nutzbar machen lässt, sondern es stellt sich darüber hinaus die Frage, ob nicht die Ausweitung des Patentschutzes auf Computerprogramme Open Source als Entwicklungs- und Vertriebsmodell grundsätzlich ebenso wie im Einzelfall unmöglich macht.

II. Software als technisches Produkt

1. Patentrecht als Alternative zum Urheberrechtsschutz

Nun lässt sich nicht von vornherein ausschließen, dass beide Schutzsysteme eine „friedliche Koexistenz" führen können. Die bisherigen Abgrenzungsversuche konzentrierten sich in Deutschland und Europa auf das Erfordernis der Technizität. Das Problem liegt jedoch darin, dass Software eine „Doppelnatur" aufweist.[2] Der Entwicklungsprozess von Software beinhaltet die Entwicklung einer Lösung für ein bestimmtes Problem mit zunehmendem Konkretisierungsgrad. Dabei entstehen verschiedene „Zwischenprodukte" bis hin zum lauffähigen „Endprodukt", dem Objektformat. Soweit dabei in einer Programmiersprache erstellte Konstrukte betroffen sind, sind diese als Ausdrucksformen geistigen Schaffens dem Urheberrecht zuzuordnen. Soweit es jedoch um die Funktionalität des „Running Code" geht, also

---

1 Richtlinie 91/250/EWG vom 14.5.1991, ABlEG L 122 vom 17.5.1991, 42; dazu Blocher/Walter, in: Walter (Hrsg.), Europäisches Urheberrecht, Wien/New York 2001, S. 111 ff.

2 Vgl. bereits Wiebe, Know-how-Schutz von Computersoftware, München 1993, S. 424 ff.; zuletzt im vertragsrechtlichen Kontext Hilty, MMR 2003, 3, 8 ff.

seine „Bestimmung", im Zusammenwirken mit einer Hardware bestimmte Aufgaben zu bewältigen, lässt sich dieser Aspekt dem Patentrecht zuordnen.[3]

Dies lässt sich etwas vereinfacht an dem Unterschied von Quellformat und Objektformat zeigen. Im Quellformat weist das Computerprogramm als Darstellungsform einen in einer Programmiersprache verfassten Text auf. Dieser linguistische Aspekt spiegelt sich in der Definition von DIN 44300 wieder:

> Eine zur Lösung einer Aufgabe vollständige Anweisung (d. h. eine in einer beliebigen Sprache abgefasste Arbeitsvorschrift, die im gegebenen Zusammenhang wie auch im Sinne der benutzten Sprache abgeschlossen ist) zusammen mit allen erforderlichen Vereinbarungen (d. h. Absprachen über in Anweisungen auftretende Sprachelemente).[4]

Im Objektformat dagegen handelt es sich um die maschinenlesbare Fassung desselben immateriellen Guts Computerprogramm, bei dem dann der technisch-funktionelle Aspekt im Vordergrund steht:

> Eine Folge von Befehlen, die nach Aufnahme in einen maschinenlesbaren Träger fähig sind zu bewirken, dass eine Maschine mit informationsverarbeitenden Fähigkeiten eine bestimmte Funktion oder Aufgabe oder ein bestimmtes Ergebnis anzeigt, ausführt oder erzielt.[5]

Wo soll nun der Schwerpunkt des Schutzes liegen? Geht es um die Förderung kreativer Tätigkeit oder um die Förderung technischer Innovation? Eigentlich liegt jeder herkömmlichen Maschine ein Algorithmus zu Grunde, der durch Interaktion mechanischer oder elektronischer Bauteile ausgeführt wird. Die Computertechnik ersetzt diese Art der Implementierung durch Arbeitsanweisungen an einen Universalcomputer. Im Mittelpunkt der zweiten industriellen Revolution, dem Informationszeitalter, steht aber auch die Ersetzung geistiger Tätigkeiten durch computergestützte Verfahren.

Der Universalrechner ist keine klassische Maschine mehr, sondern erst die Software bestimmt die Funktionalität. Sie ist eine „transklassische Maschine", oder anders ausgedrückt: „Der Algorithmus ist die Maschine".[6] Damit wird es möglich, Materie und Energie durch Information zu ersetzen. Folge ist, dass ein immer größerer Teil der Innovationstätigkeit sich auf Softwareerstellung verlagert. Dies spricht eher dafür, Software als technisches Produkt anzusehen, das in den Bereich des Patentrechts gehört.[7]

Dann ist davon auszugehen, dass sich Software nicht so fundamental von anderen technischen Gegenständen unterscheidet, dass auch dafür die Prämisse der För-

---

3 Eingehend dazu und mit dem Versuch einer „Synthese" Horns, GRUR 2001, 1, 2 ff.

4 DIN 44300 Teil 4 Nr. 4.1.9 (1988), zit. nach Haberstumpf, in: Lehmann (Hrsg.), Rechtsschutz und Verwertung von Computerprogrammen, 2. Aufl., Köln 1993, S. 78.

5 § 1 Mustervorschriften WIPO, GRUR Int. 1978, 286.

6 Troller, CR 1987, 278, 280.

7 Vgl. bereits Dryja, Looking to the Changing Nature of Software for Clues to its Protection, University of Baltimore Intellectual Property Law Journal, vol. 3, 1995, No. 2, S. 109 ff., der darauf hinweist, dass mit dem Übergang zur objektiorientierten Programmierung der Programmierer sich nicht mehr als „artist", sondern als „engineer" verstehe.

derung innovativer Entwicklungen gelten muss.[8] Allerdings gibt es in der ökonomischen Forschung keine schlüssige Antwort auf die Frage nach der Sinnhaftigkeit des Patentrechts.[9]

Einige Vorteile der Patentierung kann man hier bereits nennen. Das Patentrecht ist in vielfältiger Weise auf technische Erfindungen zugeschnitten.[10] Anders als das Urheberrecht schützt das Patentrecht das eigentlich Wertvolle bei Computerprogrammen, nämlich die Algorithmen.[11] Es ist ein effektiv auf die Technik zugeschnittenes, starkes, aber dafür zeitlich gegenüber dem Urheberrecht stärker begrenztes Schutzrecht. Es hat, und darin trifft es sich mit dem Anliegen der Open-Source-Bewegung, auch den Zweck, technisch innovative Ideen frühzeitig offen zu legen (Informationsfunktion) und damit das allgemeine Wissen zu bereichern. Es regt insoweit die innovative Tätigkeit an, um das Patent „herum" zu erfinden, also „Umgehungserfindungen" zu tätigen und damit den Substitutionswettbewerb zu fördern. Das Patent begründet keine Monopolstellung, sondern dient der Förderung des Wettbewerbs. Hinzu kommt: Patentschutz ist auch ein Schutzinstrument für kleine und mittlere Unternehmen (KMU) gegen große Unternehmen. Vielleicht wäre Microsoft mit umfassender Softwarepatentierung nicht so groß geworden, da die kleinen Mitbewerber ihre Ideen besser hätten schützen können.

Allerdings muss bereits an dieser Stelle gesagt werden, dass es stark darauf ankommt, dass die Patentansprüche nicht zu breit gefasst und nicht zu nachlässig geprüft werden, damit ein Patent nicht innovationshindernd wirkt. Ein weiterer Nachteil kommt hinzu: Wegen der hohen Schutzvoraussetzungen des Patentrechts ist nur ein kleiner Teil der Programme geschützt, also die wirklich innovativen. Daneben besteht aber auch ein Bedürfnis nach einem breiten Schutz gegen Piraterie. Dieses kann sowohl durch einen sehr eng verstandenen Urheberrechtsschutz als auch durch einen ergänzenden wettbewerbsrechtlichen Leistungsschutz gewährleistet werden.[12]

---

8 Vgl. demgegenüber Lutterbeck/Gehring/Horns, Sicherheit in der Informationstechnologie und Patentschutz für Software-Produkte – Ein Widerspruch?, Kurzgutachten, Berlin 2000, http://www.sicherheit-im-internet.de/download/Kurzgutachten-Software-patente.pdf, S. 59 f., mit der gegenteiligen Schlußfolgerung unter Berufung auf ökonomische Studien.

9 Vgl. Blind/Edler/Nack/Straus, Softwarepatente. Eine empirische Analyse aus ökonomischer und juristischer Perspektive, Heidelberg 2003, S. 103, 204; Ullrich, Grenzen des Rechtsschutzes: Technologieschutz zwischen Wettbewerbs- und Industriepolitik, in: Schricker/Dreier/Kur (Hrsg.), Geistiges Eigentum im Dienst der Innvoation, Baden-Baden 2001, S. 83, 100 ff. m. w. Nachw.; Grandstrand, The Economics and Management of Intellectual Property, Cheltenham, UK, Northampton, USA 2000; Lutterbeck/Gehring/Horns, S. 50 ff.

10 Vgl. auch die Diskussion der Vor- und Nachteile des Patentschutzes für Software bei Haase, Die Patentierbarkeit von Computerprogrammen, Hamburg 2003, S. 176 ff.

11 Diese Aussage ist allerdings abhängig von der zu Grunde gelegten Definition von Algorithmen, vgl. Wiebe, in: Spindler (Hrsg.), Rechtsfragen der Open Source Software, Köln 2004, Kap. F Rn. 40 f.; Horns, GRUR 2001 1, 5 ff.

12 Vgl. bereits Wiebe, Know-how-Schutz von Computersoftware, München 1993, S. 424 ff.

2. Geschichte der Softwarepatentierung: Ein langer Leidensweg

Die Frage, ob es sich bei softwarebezogenen Erfindungen um technische Gegenstände handelt, hat die Rechtsprechung seit 30 Jahren beschäftigt.[13] Dazu ein kleines Einführungsbeispiel.

a) Beispielsfall: „Automatische Absatzsteuerung"

Das BPatG hatte im Jahre 2000 über folgende Patentanmeldung zu entscheiden: „Verfahren zur automatischen Absatzsteuerung für eine Menge von Waren und/oder Dienstleistungen unter Verwendung eines digitalen Verarbeitungssystems, [...] gekennzeichnet durch folgende Verfahrensschritte:

- elektronisches Speichern von zeitlichen Absatzprognosedaten für mindestens einen bestimmten Abgabepreis der Waren/Dienstleistungen;
- elektronisches Erfassen aktueller Absatzdaten für die Menge von Waren/Dienstleistungen;
- elektronische Auswahl eines angepassten Abgabepreises als Funktion der Abweichung der aktuellen Absatzdaten von den Absatzprognosedaten;
- Anzeigen des ausgewählten Abgabepreises."[14]

Vereinfacht ging es dabei um einen Getränke-Automaten, bei dem programmgesteuert der Preis der Getränkedosen an die elektronisch erfasste Nachfrage angepasst wurde. Dieser Anspruch ist paradigmatisch für die technische Entwicklung. Was seit Jahrhunderten der Kaufmann hinter dem Ladentisch mit Bleistift und Kontobüchern machte, wird heute durch ein Computerprogramm übernommen und vollautomatisch durchgeführt. Das Besondere an dem Anspruch ist aber, dass nicht nur der Absatzvorgang selbst, sondern auch die betriebswirtschaftliche Regel automatisiert wird, wonach der Marktpreis sich nach Angebot und Nachfrage regelt.

b) Technische Entwicklung und Zweck des Patentrechts

Eine Erfindung als Gegenstand des Patentrechts ist eine „Lehre zum planmäßigen Einsatz beherrschbarer Naturkräfte zur Erreichung eines kausal übersehbaren Erfolges."[15] Mit dem Bezug auf den Einsatz beherrschbarer Naturkräfte ist zugleich das Technikerfordernis Teil des Erfindungsbegriffs. Eigentlich stammt das Technikerfordernis bereits aus einer Entscheidung des Reichsgerichts (RG) des Jahres 1933, in der es um die Patentfähigkeit einer Multiplikations- und Divisionstabelle in Buchform ging (RG GRUR 1933, 289).

Eine Erfindung müsse „ihrem Wesen nach" technisch sein, also mit den Mitteln von Naturkräften arbeiten, während das Rechenwerk von Tabellen dem Gebiet rein geistiger Tätigkeit angehöre und außer Betracht bleiben müsse. „Die Technik bezieht sich auf die Erscheinungswelt im Gegensatz zur Welt des Geistes; sie arbeitet nach den Lehren von Physik und Chemie" (RG GRUR 1933, 289, 290).

---

13 Eine Übersicht zur Entwicklung des Patentschutzes von Software geben Tauchert, GRUR 1999, 829 ff.; Nack, GRUR Int. 2000, 853 ff.: Kraßer, GRUR 2001, 959 ff.; Schölch, GRUR 2001, 16 f.; Anders, GRUR 2001, 555 ff.; Klopmeier, Mitt. 2002, 65 ff.; Sedlmaier, Mitt 2002, 55 ff.

14 BPatG, CR 2000 97.

15 BGH GRUR 1969, 672 – Rote Taube.

Das Patentrecht ist ein Kind der Industrialisierung. Sein Ziel war es von Anfang an, die Innovationstätigkeit zu fördern und damit technischen Fortschritt anzuregen. Technischer Fortschritt bedeutete Fortschritt der gewerblichen Güterproduktion und wirtschaftliches Wachstum. Das hieß im 19. Jahrhundert: Entwicklung neuer Maschinen und Ersetzung physischer Tätigkeit. Der Einsatz von Naturkräften, Materie und Energie, führt zu Einsparung von Arbeitskraft und/oder Kapital.

Der Siegeszug des Computers führte zur Automatisierung sowohl von Maschinen- als auch von Geistestätigkeit. Jeder herkömmlichen Maschine und jedem technischem Verfahren liegt ein Algorithmus zu Grunde, der durch Interaktion mechanischer oder elektronischer Bauteile ausgeführt wird. Die Computertechnik ersetzt diese Art der Implementierung durch Arbeitsanweisungen an einen Universalcomputer. „Der Algorithmus ist die Maschine." Im Mittelpunkt der zweiten industriellen Revolution, dem Informationszeitalter, steht aber auch die Ersetzung geistiger Tätigkeiten durch computergestützte Verfahren. Die Rechtsprechung stand daher vor der Frage, ob solche Verfahren zum Gebiet der Technik gehören und damit patentfähig sein können.

Diese Frage wirkt zunächst überraschend: ein Computer ist ein technisches Gerät und die Software steuert dieses technische Gerät: Software und Hardware gehören untrennbar zusammen. Wie kann es also Zweifel geben, dass ein softwaregestütztes Verfahren technischen Charakter hat?

Ein Beispiel soll die Problematik noch deutlicher machen. Jemand konzipiert eine Software zur Berechnung von Versicherungsprämien. Im Mittelpunkt steht dabei eine neuartige mathematische Formel, die das tatsächliche Versicherungsrisiko besonders präzise erfasst und daher sehr günstige Versicherungsprämien ermöglicht.[16] Diese Software lässt sich auf einem handelsüblichen PC ausführen. Ist das Computerprogramm patentfähig?

Dabei sind mehrere Aspekte auseinander zu halten. Der Computer ist ein technisches Gerät. Software bedarf des Computers zur Ausführung ihrer Funktionen, beide sind also nur zusammen sinnvoll einsetzbar. Das eigentlich Innovative bei diesem Fall ist die Entwicklung der Formel, deren Gegenstand nicht zum traditionellen Gebiet der Technik gehört. Deren Implementierung in eine Software ist wiederum eine handwerkliche Leistung, die von jedem Programmierer durchgeführt werden kann. Durch die Implementierung in Software wird ein traditionell nichttechnischer Gegenstand in ein „Gewand" gekleidet, das zum Bereich der Technik gehört.

In der über dreißigjährigen Geschichte der Softwarepatentierung ist eine Brandmauer nach der anderen gegen die Patentierung solcher Ansprüche gefallen. Die Entwicklung kann hier nicht nachgezeichnet werden.[17] Es bleibt jedoch festzuhalten, dass vor allem die Rechtsprechung der deutschen Gerichte, zu einem geringeren Grad auch die des EPA, durch starke Einzelfallbezogenheit und Mangel an klaren und verständlichen Regeln gekennzeichnet war. Insgesamt ist jedoch ein klarer Trend von einer auf technische Steuerprogramme und Betriebssysteme beschränkten Patentierung hin zu einer Auffassung, die nahezu jede Softwareimplementierung als technisch ansieht, zu konstatieren. Der BGH hat zuletzt festgestellt, dass, wenn

16 Vgl. Nack, GRUR Int. 2000, 853; EPA, CRi 2001, 18.

17 S. die Nachw. o. Fußn. 13.

die industrielle Entwicklungstätigkeit zunehmend automatisiert durchgeführt würde, auch der Technikbegriff angepasst werden müsse. Entsprechend könne auch auf das Erfordernis des unmittelbaren Einsatzes beherrschbarer Naturkräfte verzichtet werden.[18]

Während der Ausschlusstatbestand für Computerprogramme als solche nach § 1 Abs. 2 Nr. 3, Abs. 3 PatG unklar ist und kaum praktische Bedeutung erlangt hat,[19] wirft dies vor allem Probleme hinsichtlich der Abgrenzungsfunktion des Ausschlusstatbestands für Geschäftsmethoden auf. Die vom BGH getroffene – eigentlich selbstverständliche – Feststellung, dass der Technikbegriff nicht statisch sei, ist hierbei von großer Bedeutung.[20] Wenn denn das (im Gesetz gar nicht explizit enthaltene, aber gewohnheitsrechtlich anerkannte) Technikerfordernis eine Abgrenzungsfunktion gegenüber nicht für schutzwürdig erachteten geistigen Leistungen hat,[21] so ist diese nach Sinn und Zweck des Patentrechts zu bestimmen. Diese muss natürlich auch Veränderungen des Normbereichs berücksichtigen. Dazu gehört nicht nur die technische Entwicklung, sondern auch die damit zusammenhängenden gesellschaftlichen Entwicklungen. So stellte der BGH in der „Rote Taube"-Entscheidung[22] fest, dass die zum Erlass des PatG 1877 herrschende Auffassung über patentfähige Erfindungen nicht mehr maßgebend sein könne, da sich „inzwischen Naturwissenschaft und Technik erheblich gewandelt haben." [23]

Ein modernes Technikverständnis umfasst nicht nur die Verlagerung herkömmlicher Maschinentechnik auf computergesteuerte Verfahren, sondern auch die Verlagerung bisheriger gedanklicher Arbeit auf Computer – und damit auf technische Verfahren. Hier findet Innovation statt und ist die Quelle von Wirtschaftswachstum. Solange es sich um eine software- oder hardwarebezogene Anspruchsformulierung handelt, bewegen wir uns im Bereich der Technik.

Sorgfältig zu prüfen sind dann die Merkmale Neuheit und erfinderische Tätigkeit. Diese weiteren Anforderungen schaffen Freiräume für die ganz alltäglichen Weiterentwicklungen. In diesem Bereich sind auch die Sünden der amerikanischen Praxis angesiedelt, die häufig bei sehr breiten Ansprüchen nicht hinreichend Neuheit und erfinderische Tätigkeit gewürdigt hat. So wurde die Unterlassungsverfügung für den „one-click-process" von Microsoft wieder aufgehoben, da bereits früher solche Verfahren genutzt worden seien.

c) Richtlinienvorschlag der EU-Kommission

Unter diesen Vorzeichen war der ursprüngliche Richtlinienvorschlag eine echte Befreiung.[24] Der Vorschlag ging davon aus, dass zunächst alle Computerprogramme

18 BGHZ 143, 255 – Logikverifikation.

19 Vgl. Bittner, in: Bröcker/Czychowski/Schäfer (Hrsg.), Praxishandbuch Geistiges Eigentum im Internet, München 2003, S. 679, 686; Wiebe, in: Spindler (Hrsg.), Rechtsfragen bei Open Source, Köln 2004, Kap. F, Rn. 23 ff.

20 Vgl. Hössle, Der nicht-statische Technikbegriff, Mitt. 2000, 343 ff.

21 BGH GRUR 1992, 36 – Chinesische Schriftzeichen.

22 Im Fall „Rote Taube" ging es um die Erweiterung des Naturkräfte-Begriffes von der Physik/Chemie auf die Biologie.

23 BGH GRUR 1969, 672.

24 Vorschlag für eine Richtlinie des EuropäischenParlaments und des Rates über die Patentierbarkeit computerimplementierter Erfindungen, KOM(2002) 92 endg. v. 20.2.2002, ABlEG C 2002/151 E

technischen Charakter aufweisen, da sie auf einem Computer ablaufen (Art. 3). Im Rahmen des Merkmals der erfinderischen Tätigkeit wurde dann aber ein „technischer Beitrag" gefordert (Art. 4 Nr. 2). Nur dieser technische Beitrag sollte bei der Prüfung der erfinderischen Tätigkeit berücksichtigt werden. Damit würde auf der einen Seite die unglückselige Verquickung des Technikerfordernisses mit der Frage, worin der innovative Beitrag der Erfindung liegt („Kerntheorie"), aufgehoben. Zum anderen würde am Technikerfordernis festgehalten, dieses aber nicht-statisch verstanden, sodass etwa die Notwendigkeit, im Vorfeld technische Überlegungen anzustellen bereits ausreichen kann.

Zur Illustration kann noch einmal die EPA-Entscheidung „Steuerung eines Pensionssystems" dienen.[25] Dort wurde die softwareimplementierte Ausführung eines Verfahrens zur Berechnung von Versicherungsbeiträgen beansprucht. Die Automatisierung stellte einen technischen Vorgang dar, war aber keine besondere erfinderische Leistung. Demgegenüber war die finanzmathematische Formel neuartig, aber beinhaltete nicht die Lösung einer technischen Aufgabe. Da im technischen Bereich keine erfinderische Leistung gegeben war, wurde die Patentfähigkeit vom EPA abgelehnt. Die EPA-Entscheidung steht insoweit im Einklang mit dem Ansatz des Richtlinienvorschlags.

Damit wären der Patentierung geschäftsbezogener Ansprüche Grenzen gesetzt, und dies war auch im ersten Entwurf die Absicht der Kommission. Nur eine technische Innovation trägt zur Patentfähigkeit bei, nicht aber eine innovative Geschäftsmethode selbst. Bei der Bestimmung der erfinderischen Tätigkeit wird auf das Wissen und Können eines Durchschnittsfachmanns in dem jeweiligen Gebiet abgestellt (§ 4 PatG). Fachmann ist nicht der Ökonom, sondern der Programmierer bzw. Wirtschaftsinformatiker. Dem Fachmann wird quasi fiktiv die Kenntnis der nichttechnischen Innovationen zugerechnet, sodass diese nicht mehr zur Begründung der Patentfähigkeit beitragen können. Die routinemäßige Softwareimplementierung einer guten Geschäftsidee ist für den Programmierer handwerkliche Tätigkeit. Das Patentamt muss keine Ökonomen oder Sprachwissenschaftler als Prüfer einstellen.

Diese auf den ersten Blick klare Abgrenzung ist natürlich schwierig in Grenzbereichen. Die computerimplementierte „Fassung" einer Geschäftsidee unterscheidet sich jedoch von der herkömmlichen Ausführungsweise. Je mehr die Ausführung der Geschäftsidee auch in ihren nichttechnischen Bestandteilen durch die Computerausführung beeinflusst und bestimmt wird, desto enger ist auch der Zusammenhang zwischen Geschäftsmethode und technischer Ausführung auf dem Rechner.

Als Beispiel lässt sich der Sachverhalt der jüngsten BGH-Entscheidung „Suche fehlerhafter Zeichenketten" anführen.[26] Dabei ging es um ein neuartiges Verfahren zur Fehlerkorrektur in digital gespeicherten Texten. Diese erfolgt nicht auf der Grundlage eines Vergleichs mit abgespeicherten korrekten Wörtern, wie wir es etwa vom Word-Programm kennen, sondern durch eine statistische Analyse des Textes. Bei einem fehlerfreien Wort werden einzelne Buchstaben ausgelassen oder vertauscht und dann durch Vergleich der Auftretenswahrscheinlichkeit mögliche feh-

v. 25. 6. 2002, 129. Dazu Sedlmaier, Mitt. 2002, 97 ff.; Röttinger, CR 2002, 616 ff.

25 EPA, CRi 2001, 18.

26 BGH GRUR Int. 2002, 323.

lerhafte Worte ermittelt. Der Vorteil eines solchen Verfahrens ist ein sehr geringer Speicherplatzbedarf. Ein solches Verfahren würde nie jemand mit Papier und Bleistift durchführen. Es ist so stark durch Einsatz der Digitaltechnik bedingt und damit verknüpft, dass es selbst als technisch anzusehen ist.[27]

Die Geschäftsmethode wird aber durch eine Patentierung nur in ihrer von einer technischen Implementierung abhängigen Form monopolisiert. Insoweit unterscheidet sich dieser Fall von der Entwicklung einer neuartigen finanzmathematischen Methode zur Berechnung von Versicherungsprämien die nur computerimplementiert ausgeführt wird, aber auch mit Papier und Bleistift ausgeführt werden könnte. Sie unterscheidet sich auch von dem eingangs angeführten Getränkeautomaten. Bei diesem entsprach die technische Umsetzung der betriebswirtschaftlichen Regel dem „herkömmlichen Rüstzeug des Fachmanns" und war daher nicht erfinderisch.[28]

Der Ansatz des Richtlinienvorschlags ist nunmehr durch das Europäische Parlament, nicht zuletzt auf Druck der Open-Source-Gemeinde, fast in sein Gegenteil verkehrt worden.

d) Entschließung des Europäischen Parlaments

In der Entschließung[29] wird bereits in Art. 4a und Erwägungsgrund 13a abweichend vom bisherigen Erwägungsgrund 13 festgestellt, dass die Computerimplementierung allein keinen „technischen Beitrag" leistet. Damit sollten computerimplementierte Erfindungen nicht mehr grundsätzlich als technisch angesehen werden.

Die Definition des „technischen Beitrags" in Art. 2b) stellt klar, dass es dabei um das Technizitätserfordernis geht.[30] Dabei wird auch die Einbeziehung von Information in den Bereich der Naturkräfte ebenso abgelehnt, wie eine Beschränkung des Technikbegriffs auf die Beherrschung von Naturkräften zur Beherrschung der physikalischen Wirkungen der Informationsverarbeitung festgelegt wird. Der vom BGH befürworteten Weiterentwicklung des Technikbegriffs wird damit ein Riegel vorgeschoben, mit vielleicht unübersehbaren Konsequenzen für die Zukunft des Patentrechts allgemein.

Zusätzlich wird nach Art 3a die Patentierbarkeit im Hinblick auf Methode und Ergebnis auf „industrielle Anwendungen" beschränkt, worunter nach Art. 2 bb „die automatisierte Herstellung materieller Güter" zu verstehen ist. Zu dieser Beschränkung auf die Industrie des 19. Jahrhunderts passt auch, dass in Art. 3a zusätzlich festgestellt wird, dass Innovationen im Bereich der Datenverarbeitung nicht patentierbar sind.[31] Nicht einmal die Einsparung von Ressourcen in einem Datenverar-

---

27 So auch Anders, GRUR 2001, 559. Die gleiche Situation ergibt sich bei Reverse Auctions. Wer teilt allen Einwohnern per Brief oder Telefon seine Bereitschaft mit, für einen Produkt einen bestimmten Preis zu zahlen. Wie kann ein potenzieller Verkäufer bei einem Angebot in der Tageszeitung sicher sein, dass er auch zum Zuge kommt, vgl. Anders, a.a. O.

28 Vgl. BPatG, CR 2000 97, 99.

29 P5_TA-PROV(2003)0402, Legislative Entschließung des Europäischen Parlaments zu dem Vorschlag für eine Richtlinie des Europäischen Parlaments und des Rates über die Patentierbarkeit computerimplementierter Erfindungen (KOM(2002)92 – Cs-0082/2002-2002/0047(COD)).

30 Vgl.auch die neue Definition „computerimplementierter Erfindungen" in Art. 2a).

31 Deren Patentierbarkeit war in der bisherigen Diskussion eigentlich bereits Konsens, vgl. Ohly, CR 2001, 809, 815.

beitungssystem gilt dann als technischer Beitrag (Art. 4b). Damit geht die Entschließung zurück auf einen Zustand vor dem Beschluss „Seitenpuffer" des BGH aus dem Jahre 1991.[32] Erwartungsgemäß werden auch Ansprüche auf ein auf einem Datenträger gespeichertes Programm allein nach Art. 5(1a) ausgeschlossen.

Zu diesem engen Technikverständnis gehört auch, dass nach Erwägungsgrund 13a eine computerimplementierte Geschäftsmethode grundsätzlich nicht-technisch sein soll. Am obigen Beispiel des Falls „Suche fehlerhafter Zeichenketten" sollte gezeigt werden, dass es sehr wohl einen Unterschied macht, ob ein Verfahren mit Papier und Bleistift oder mittels eines Computers ausgeführt wird. Obwohl man insoweit auch einen technischen Beitrag zum Stand der Technik annehmen könnte, erscheint eine solche Interpretation nach der Intention der Entschließung als eher unwahrscheinlich.

Wenn in Art. 4 für das Kriterium der erfinderischen Tätigkeit ein „technischer Beitrag" gefordert wird, so entspricht das zunächst dem hier vertretenen Standpunkt. Allerdings wird in Erwägungsgrund 11 für die erfinderische Tätigkeit „zusätzlich ein neuer technischer Beitrag zum Stand der Technik" gefordert. Hier werden die Merkmale der Erfindungshöhe und der Neuheit vermischt, was sehr an die frühere „Kerntheorie" des BGH erinnert. Auch in Bezug auf die Erfindungshöhe wird in Art. 4 (3) eine Abkehr von einer Gesamtbetrachtung hin zu einer Konzentration auf die technischen Merkmale vollzogen.

Art. 6 eröffnet scheinbar ein verfassungsrechtlich problematisches Primat des Urheberrechtsschutzes gegenüber dem Patentschutz, indem die in Art. 5, 6 Software-RL (§§69d, e UrhG) festgelegten Schranken bei kollidierendem Patentschutz vorrangig sein sollen. Allerdings kann man die Voraussetzung des „rechtmäßigen Erwerbers/Nutzers" in §69d UrhG auch so verstehen, dass dieser das Programm ohne Verstoß gegen das Patentrecht erworben hat, sodass die Konfliktlage dann weitestgehend entschärft ist. Wie die Ausnahme vom Schutzumfang „für einen bedeutsamen Zweck" in Art. 6a verfassungskonform zu handhaben ist, erscheint ebenfalls unklar.

Was bleibt an Positivem zu berichten? Zu begrüßen ist die Klarstellung in Erwägungsgrund 13c, wonach ein Algorithmus grundsätzlich nichttechnischer Natur ist und nur bei Anwendung zur Lösung eines technischen Problems patentierbar ist und nur in diesem Kontext geschützt wird. Allerdings hängt die Reichweite dieser Freistellung entscheidend vom Begriffsverständnis ab.

Zu begrüßen ist ebenfalls der Hinweis darauf, dass die Schutzvoraussetzungen der Neuheit und erfinderischen Tätigkeit genau zu beachten sind.[33] Sollte es danach überhaupt noch computerimplementierte Erfindungen geben können, so ist die Forderung nach der Veröffentlichung einer „gut dokumentierte(n) Referenzimplementierung eines solchen Programms als Teil der Beschreibung ohne einschränkende Lizenzbedingungen" nach Art. 5 (1d) zu fordern.

Insgesamt kann man konstatieren, dass die Entschließung wenig Spielraum für computerimplementierte Erfindungen lässt und die Rechtsentwicklung um mindestens zehn Jahre zurückwerfen wird. Es wird wieder das Problem der Abgrenzung

[32] BGH CR 1991, 658.
[33] Vgl. Erw. grd. 18a.

technischer von nichttechnischen Programmen auftauchen, da die Richtlinie das Bedürfnis nach Patentierung in diesem Bereich nicht einfach vom Tisch wischen kann. Entgegen der ausdrücklichen Intention der Entschließung in Erwägungsgrund 5a wäre damit wieder große Rechtsunsicherheit geschaffen, die eine der größten Hindernisse für innovative Tätigkeit darstellt. Die Auswirkungen auf den Schutzzweck des Patentrechts sind unabsehbar.

III. Patentschutz und Open-Source-Software: Konfliktlage

Neben der Problematik der Geschäftsmethoden hat sich die Diskussion zur Frage Patentschutz und Software vor allem auf den Open-Source-Bereich zugespitzt. Geht man davon aus, dass es sich bei Open Source um ein alternatives Entwicklungs- und Vertriebsmodell handelt,[34] dass das Ausschließlichkeitsprinzip des Immaterialgüterrechts grundsätzlich in Frage stellt, dann kann nur eine völlige Ablehnung eines Patentschutzes für diesen Bereich die Folge sein.[35] Anlass genug, einmal näher zu untersuchen, welche Auswirkungen ein Patentschutz für Software auf Entwickler und Nutzer in diesem Bereich haben kann.[36]

1. Grundsätzliche Bedenken gegen den Patentschutz

Nachdem oben die Vorteile des Patentschutzes herausgestellt wurden, sollen auch die immer wieder vorgebrachten Befürchtungen, insbesondere im Hinblick auf Open-Source-Entwickler, nicht verschwiegen werden. Mit der Erfassung von Algorithmen im konkreten Sinne ist die Befürchtung verbunden, dass die Software-Entwicklungstätigkeit durch ein „dichtes Gestrüpp von patentierten Algorithmen“[37] erschwert werde, die die Funktionalität von Programmen in vielen Bereichen abdecken und damit das Entwickeln von Programmen in diesem Bereich lizenzpflichtig machen könnten. Diese Gefahr besteht aber nur bei zu breiter Patentierung, wie sie vor allem in den USA vorgekommen ist. Bei sorgfältiger Patentprüfung und gut dokumentiertem Stand der Technik dürfte die Gefahr gering sein, dass die Programmierung ernsthaft behindert wird, auch wenn Programmierer oft ins Feld führen, dass ihre Tätigkeit sich von herkömmlicher technischer Entwicklungstätigkeit grundsätzlich unterscheide.

Ein weiteres Problem besteht darin, dass eine Recherche auf Grund der vielfältigen Verletzungsformen schwierig ist und das daraus entstehende Patentverletzungsrisiko vor allem für kleinere und mittlere Unternehmen schwer zu tragen sein kann.

---

34 Lutterbeck/Gehring/Horns, Sicherheit in der Informationstechnologie und Patentschutz für Software-Produkte – Ein Widerspruch?, Kurzgutachten, Berlin 2000, http://www.sicherheit-im-internet.de/download/Kurzgutachten-Software-patente.pdf, S. 61 ff.

35 Vgl. Lutterbeck, Jefferson revisited/revised: Demokratische Technologie und Softwarepatente sind ein Widerspruch!, http://ig.cs.tu-berlin.de/bl/088/Lutterbeck-FIFFJefferson-2003.pdf.

36 Vgl. auch Gehring/Lutterbeck, Software-Patente im Spiegel von Softwareentwicklung und Open Source Software, in: Taeger/Wiebe (Hrsg.), Recht-Wirtschaft-Informatik, Festschrift für Wolfgang Kilian, Baden-Baden 2004, S. 301 ff.; Wiebe, in: Spindler (Hrsg.), Rechtsfragen bei Open Source, Köln 2004, Kap. F Rn. 43 ff.

37 Metzger, Risiko Softwarepatente?, http://www.ifrOSS.de/ifrOSS_html/art11.pdf, S. 2. Vgl. auch Lutterbeck, in: Taeger/Wiebe (Hrsg.), S. 301, 307 ff., mit Hinweis auf die „Granularität“ von Software und die zunehmende Erfassung von Basistechnologie.

Bei den Rechercheproblemen handelt es sich jedoch um typische Anfangsprobleme bei neuen Technologien. Zudem ist es Open-Source-Entwicklern als Abwehrstrategie möglich, durch Einrichtung von Dokumentationsstellen, in denen Softwareentwickler ihre Arbeitsergebnisse mit Zeitstempel registrieren und der Öffentlichkeit zugänglich machen können, den Stand der Technik zu bereichern und damit der Erteilung von Patenten auf solche Software entgegenzuwirken.[38]

Bei Open Source besteht auf Grund der Verfügbarkeit des Quellcodes der Nachteil, dass die Feststellung einer Patentverletzung einfacher wird und damit die Rechtsverfolgung durch „proprietär" ausgerichtete Unternehmen erleichtert wird, die selbst ihre Programme häufig nur im Objektformat verbreiten, dessen Rückentwicklung durch das Urheberrecht verhindert wird. Andererseits ist wegen der funktionalen Ausrichtung des Patentschutzes häufig die Verfügbarkeit des Quellformats noch nicht ausreichend, um eine Patentverletzung festzustellen.

Eine weiterer Nachteil gegenüber „proprietären" Anbietern ist die Schwierigkeit einer aktiven Patentrechtspolitik („konzeptionelle Asymmetrie" des OSS-Modells).[39] Patentrecherche und Patentstreitigkeiten sind kostenintensiv und von den meist kleinen oder mittelständischen Entwicklern und Distributoren finanziell schwerer durchzustehen. „Kreuzlizenzen" sind eine häufige Form der Patentverwertung: mit eigenen Lizenzen als „Währung" erwirbt man Zugang zu Technologien der Mitbewerber. Dieser Weg ist Open-Source-Entwicklern weitgehend verwehrt.

Schließlich wird auch ins Feld geführt, dass der Patentschutz zu einer Blockade von Standards führe, indem kleinere, nicht zu einem Patentpool gehörende Unternehmen am Zugang gehindert werden. Praktisch hat sich das aber bisher kaum als Problem erwiesen.[40]

2. Verletzung bestehender Patente durch Open-Source-Entwickler

Auf dem Hintergrund dieser allgemeinen Bedenken stellt sich insbesondere die Frage, inwieweit die Softwareentwicklung selbst durch einen Patentschutz behindert wird. Dazu sind die verschiedenen Konfliktsituationen näher zu beleuchten. Im Vordergrund steht zunächst die Situation, dass ein Open-Source-Entwickler Teile einer patentrechtlich geschützten softwareimplementierten Erfindung nutzt und auch weiterverbreitet. Hier stellt sich die Frage, inwieweit der Schutzbereich des „proprietären" Programms berührt ist.

a) Patentverletzung §§ 9, 10 PatG

Zumeist ist der softwarebezogene Patentanspruch als Verfahrenserfindung gefasst, sodass sich eine Patentverletzung auf die Anwendung des Verfahrens sowie das Anbieten zur Anwendung beschränkt. Wegen der Ausrichtung auf die Funktionalität geht es bei der Patentverletzung nicht um die Ähnlichkeit der Programmbefehle, sondern um eine Ähnlichkeit der „dynamischen" Funktionsabläufe.[41] Da man sich nicht allein auf die Quellprogramme stützen kann, erschwert dies die Recherche

[38] Vgl. Lutterbeck/Gehring/Horns, S. 11 f.; Jaeger/ Metzger, Open Source Software, München 2002, S. 112.

[39] Horns, www.jurpc.de/aufsatz/20000223.htm, Abs. 55.

[40] Vgl. Haase, S. 178.

[41] Vgl. auch Heide, CR 2003, 165, 167.

des Stands der Technik.[42] Gerade auf Grund der Kennzeichnung von Ansprüchen durch funktionale Merkmale kann der Schutzumfang besonders durch eine übermäßig breite Fassung der Ansprüche erweitert werden.

Neuerdings gewährt die Rechtsprechung auch Erzeugnispatente auf das Computerprogramm selbst.[43] Der Anspruch ist dann auf ein digitales Speichermedium gerichtet, auf dem das Programm gespeichert ist. Dies soll jedenfalls insoweit patentierbar sein, als es als Mittel zur Ausführung eines Verfahrens dient und dieses Verfahren technischen Charakter aufweist.[44] Dann stellt bereits die Herstellung, das Anbieten und Inverkehrbringen, auch über Internet, eine Rechtsverletzung dar, was eine erhebliche Ausweitung des Schutzes bedeutet und auch die Weiterverbreitung von Open-Source-Software erschwert.[45]

Allerdings kommt es auf die Fassung der Ansprüche an, ob das Anbieten der Software sich nur auf einen Teil der Erfindung bezieht und damit in den Bereich der mittelbaren Patentverletzung gem. §10 PatG fällt.[46] Auch bei einer Verfahrenserfindung kann das Computerprogramm ein Teilsystem darstellen, das ein wesentliches Element des geschützten Verfahrens konstituiert.[47] Für eine Rechtsverletzung bedarf es hier besonderer subjektiver Voraussetzungen.

Aus einer Patentverletzung können sich Unterlassungs- und Schadensersatzansprüche ergeben. Auch wenn eine Haftung auf Schadensersatz Verschulden voraussetzt und von einem einfachen Entwickler keine breite Patentrecherche verlangt werden kann,[48] ist doch zu beachten dass die einschlägigen Patente in Programmiererkreisen zumindest soweit bekannt sein können, dass Anlass zu weitergehender Recherche besteht.

b) Ausnahmetatbestände § 11 PatG

Selbst wenn die Entwicklung und Verbreitung von Open-Source-Software ein bestehendes Patent verletzen sollte, kann es sein, dass eine der Ausnahmen vom Schutzumfang greift. Nach §11 Nr. 1 PatG gehören dazu Handlungen im privaten Bereich zu nichtgewerblichen Zwecken. Hierunter fällt also die private Nutzung von Open-Source-Software. Beim Austausch oder Vertrieb von Software über das Internet wird aber wohl die Grenze des privaten Bereichs überschritten.[49] Bei einer Verfahrenserfindung bleibt zu beachten, dass ein Anbieten zu erlaubten Zwecken nach §9 Abs. 1 Nr. 2 PatG möglich bleibt. Dies wäre eigentlich bei einer Weitergabe an einen Nutzer zu dessen privatem Gebrauch gegeben. Allerdings werden dem

42 Vgl. Horns, www.jurpc.de/aufsatz/20000223.htm, Abs. 53.

43 EPA , CR 2000, 91, 93 – Computerprogrammprodukt/IBM; BGH, GRUR Int. 2002, 323 – Suche fehlerhafter Zeichenketten. S.o. Fußn. 20 f.

44 BGH, Beschluß v. 17.10.2001 – X ZB 16/00, GRUR Int. 2002, 323 – Suche fehlerhafter Zeichenketten. Kritisch dazu Vorschlag für eine Richtlinie des EuropäischenParlaments und des Rates über die Patentierbarkeit computerimplementierter Erfindungen, KOM(2002) 92 endg. v. 20.2.2002, Begründung, S. 16.

45 Vgl. Esslinger/Betten, CR 2000, 18, 20.

46 Vgl. Heide, CR 2003, 165, 166.

47 Vgl. Lutterbeck/Gehring/Horns, S. 98 ff., die als Beispiel das separate Anbieten eines Speicherbausteins für eine Kraftfahrzeugbremsensteuerung nennen.

48 So auch Metzger, http://www.ifrOSS.de/ifrOSS_html/art11.pdf, S. 2.

49 Vgl. auch Esslinger/Betten, CR 2000, 18, 21.

Nutzer keine Weitergabebeschränkungen auferlegt, sodass eine Eingrenzung auf den privaten Gebrauch nicht gewährleistet werden kann, und damit wohl doch der Verletzungstatbestand erfüllt wäre.[50]

Freigestellt werden durch §11 Nr. 2 PatG weiterhin Handlungen zu Versuchszwecken, die sich auf den Gegenstand der patentierten Erfindung beziehen. Obwohl die Rechtsprechung die Grenzen dieses Ausnahmetatbestands relativ eng zieht, dürfte die Open-Source-Entwicklung darunter fallen, wenn man das Prinzip darin sieht, dass ständig Verbesserungen an der geschützten Software vorgenommen werden. Diese Schranke dient der Forschung und der technischen Innovation. Allerdings darf eine resultierende Entwicklung nicht in den Schutzbereich bestehender Patente eingreifen. Zu beachten ist dabei auch, dass bei parallel bestehendem Urheberrecht ein „reverse engineering" nur in den engen Grenzen von §69e UrhG zulässig ist.[51]

3. Schutz gegen Patentanmeldungen durch „proprietär" ausgerichtete Softwareunternehmen

Möglich ist, dass „proprietär" ausgerichtete Unternehmen Patente anmelden, deren wesentliche Bestandteile aus Open-Source-Software bestehen. Eine zentrale Abwehrstrategie der Open-Source-Entwickler besteht hier darin, dass die Software möglichst schnell veröffentlicht wird, um somit die Neuheit einer Erfindung nach §3 PatG auszuschließen. Dabei reicht eine Veröffentlichung irgendwo auf der Welt bereits aus, sodass auch eine Zugänglichmachung des Quellcodes über Internet ausreicht, wenn es nicht um private E-Mails geht. Um Beweisschwierigkeiten zu vermeiden, erscheint die Einrichtung einer Dokumentationsstelle sinnvoll.[52]

Einen Schutz gegen einen Patentinhaber bietet auch die Berufung auf das Vorbenutzungsrecht des §12 PatG. Das setzt voraus, dass der Open-Source-Entwickler die Erfindung erkannt hat und diese vor der Patentanmeldung bereits benutzt oder die zur baldigen Anwendung notwendigen Maßnahmen getroffen hat. Hat der Open-Source-Entwickler die Kenntnis vom späteren Schutzrechtsinhaber erlangt, entsteht ein Vorbenutzungsrecht nicht, wenn er sich die Kenntnis widerrechtlich verschafft hat.[53] Ein Vorbenutzungsrecht verleiht einen fortbestehenden Anspruch auf Benutzung der Erfindung im Inland. Eine entscheidende Einschränkung ergibt sich jedoch aus der Beschränkung auf den eigenen Betrieb. Eine solche betriebsbezogene Verwertung wird für das Open-Source-Prinzip nur dann anzunehmen sein, wenn die Entwicklung innerhalb fest organisierter Communities erfolgt und neu entwickelte Versionen vor ihrer Verbreitung zunächst durch den „Kopf" der Organisation genehmigt werden müssen.

Relevant erscheint auch der umgekehrte Fall, nämlich dass der „proprietäre" Entwickler die patentierte Lehre einem Open-Source-Programm entnommen und dann selbst angemeldet hat. Nach §6 PatG steht das Recht auf das Patent dem Erfinder zu. Hat der Anmelder den wesentlichen Inhalt der Erfindung Beschreibun-

50 Vgl. auch Jaeger/Metzger, S. 119.
51 Vgl. auch Jaeger/Metzger, S. 117.
52 Vgl. Jaeger/Metzger, S. 114 ff.
53 Vgl. BGH GRUR 1964, 675.

gen oder Unterlagen des Erfinders ohne dessen Einwilligung entnommen, kann das Patent auf Einspruch hin widerrufen werden (§21 Abs. 1 Nr. 3 PatG).[54] Eine bloße Mitteilung oder Überlassung zum Nachbau beinhaltet noch keine Einwilligung zur Patentanmeldung.[55] Auch hier hat ein Open-Source-Entwickler also ein Abwehrmittel in der Hand, vorausgesetzt, die Open-Source-Software und die später angemeldete Erfindung stimmen in ihrem wesentlichen Inhalt überein.

4. Schutz gegen Patentanmeldungen durch Open-Source-Entwickler

Schließlich droht der Open-Source-Gemeinde auch „Gefahr" aus den eigenen Reihen, wie der Fall „RTLinux" gezeigt hat.[56] Kann ein Patentanmelder einer Open-Source-Software mittels der Lizenz verpflichtet werden, die Software frei zu lizenzieren? Die Reichweite etwa der GNU-Lizenz ist davon abhängig, inwieweit Weiterentwicklungen, die möglicherweise patentfähig sind, im urheberrechtlichen Schutzbereich liegen.[57] Nur wenn urheberrechtlich relevante Nutzungshandlungen bei der Entwicklung des Derivats erfolgen, ergibt sich aus der GNU-GPL eine Verpflichtung, das Derivat auch patentrechtlich unter die Bedingungen der Lizenz zu stellen.[58] Die „Open RTLinux Patent Licence"[59] erlaubt die lizenzgebührenfreie Nutzung des Patents entsprechend auch nur, wenn GPL-Software eingesetzt wird. In anderen Fällen ist eine „proprietäre" Lizenz einzuholen.[60]

IV. Lösungsvorschläge

Verbleibt somit ein Konfliktpotential, so werden verschiedene Wege zur Lösung diskutiert.

1. Schutzzweckbezogene Auslegung des Patentgesetzes

Ein wichtiger Eckpfeiler dafür, dass das Patentrecht wirklich die ihm zugedachte Wirkung der Förderung der Innovation erfüllen kann, ist die korrekte Anwendung der gesetzlichen Voraussetzungen. Dazu gehört, dass hohe Anforderungen an das Vorliegen der Erfindungshöhe und Neuheit zu stellen sind.[61] Weiterhin ist zu beachten, dass auch im Patentrecht der Grundsatz der Schutzfreiheit allgemeiner Lehren und Methoden gilt (§1 Abs. 2 Nr. 1 PatG). Richtig angewendet, müssen danach auch Algorithmen einer bestimmten Abstraktionsstufe frei bleiben.[62]

Möglich ist auch eine Begrenzung des Schutzumfangs softwarebezogener Patente. Zwar entspricht die Einbeziehung äquivalenter Lösungen ständiger Rechtspre-

54 Vgl. auch Jaeger/Metzger, S. 116.
55 Vgl. Benkard, PatG, § 9 Rn. 9.
56 Vgl. Jaeger/Metzger, S. 121.
57 Vgl. dazu Wuermeling/Deike, CR 2003, 87, 88 ff.; Jaeger/Metzger, S. 123 ff.; Jaeger, in: Wiebe/Leupold (Hrsg.), Recht der elektronischen Datenbanken, Heidelberg 2002, III-B Rn. 79 ff.
58 Jaeger/Metzger, S. 124, meinen darüber hinaus, dass § 2b) GPL auch Patente erfasst, wenn zugunsten freier Entwicklergruppen bereits Patente bestehen, was aber kaum praktisch werden dürfte.
59 Vgl. http://www.fsmlabs.com/products/rtlinuxpro/rtlinux_patent.html.
60 So auch Jaeger/Metzger, S. 125 f., mit Hinweis auf die 2. Version der Open RTLinux Patentlizenz.
61 Gerade die Neuheit der in den USA erteilten Patente wird immer wieder bezweifelt, vgl. etwa zum RTLinux-patent Jaeger/Metzger, S. 121 Fußn. 517.
62 Vgl. Wiebe, in: Spindler (Hrsg.), Rechtsfragen bei Open Source, Kap. F Rn. 40 ff.

chung des BGH und hat eine Grundlage auch im Auslegungsprotokoll zu Art. 69 EPÜ. Allerdings sollten die Gerichte bei der Bestimmung der technischen Gleichwirkung nicht zu großzügig vorgehen und die Besonderheiten der Informatik im Sinne handhabbarer Abgrenzungen berücksichtigen. Dadurch könnten in Kombination mit den Ausnahmen von § 11 Nr. 1, 2 PatG Freiräume geschaffen werden.

2. Quelltext und Waffengleichheit

Vorgeschlagen ist weiterhin, den Vertrieb von Software im Quelltext vom Schutzumfang des Patentrechts auszunehmen.[63] Damit wäre ein Teil des Open-Source-Entwicklungsmodells von der Behinderung durch den Patentschutz befreit.

Ein interessanter Vorschlag ist auch, „Waffengleichheit" durch eine Verpflichtung „proprietärer" Software-Patentanmelder zur Hinterlegung eines Programmlistings herzustellen.[64] Damit hätten die Open-Source-Entwickler noch besseren Zugang zum patentierten technischen Wissen. Andererseits besteht natürlich ein gravierendes Geheimhaltungsinteresse, soweit dies im Hinblick auf den Offenbarungsgrundsatz möglich ist.[65] Ein interessanter Weg kann jedoch in der in Art. 5 (1d) der Entschließung des Europäischen Parlaments vorgeschlagenen „Referenzimplementierung" liegen.[66]

3. Anwendung der Zwangslizenzregelung (§ 24 PatG)

Die Möglichkeit der Erteilung einer Zwangslizenz ist eher ein seltener Ausnahmefall und das dafür notwendige öffentliche Interesse wurde bisher nur sehr restriktiv angenommen. Es ist eine auf den Einzelfall zugeschnittene Ausnahmeregelung, muss durch Klage (§ 81 Abs. 1 PatG) durchgesetzt werden und ist insoweit praktisch zur Absicherung des Open-Source-Modells nicht geeignet.

4. Patent Pool – die kollektive Lösung

In Anlehnung an die Patentrechtspolitik größerer Unternehmen wird für Open-Source-Software die Bildung von Patentpools vorgeschlagen, die der Abwehr von Patentverletzungen und der Kreuzlizenzierung mit einer breiteren Basis dienen könnten.[67] Voraussetzung eines solchen Pools ist aber der Erwerb eigener Patente. Dies ist allerdings nach der derzeitigen Gesetzeslage im Hinblick auf die Funktionsweise des Open-Source-Modells weitgehend ausgeschlossen. Zumindest eine gewisse Erleichterung bringen Modelle des „Dual Licensing", wie von RTLinux praktiziert, sowie der Eclipse-Lizenz von IBM, die in einem Projekt entwickelte Software von Patentansprüchen gegen die entwickelte Software freistellt.[68]

---

63 Vgl. Lutterbeck/Gehring/Horns , S. 132: „Herstellen, Anbieten, In-Verkehr-bringen, Besitzen oder Einführen von Software im Quellcode".

64 Vgl. Haase, S. 175.

65 Vgl. Stobbs, Software Patents, New York 1995, S. 170, wonach der Source Code für Zwecke der Offenbarung einerseits zu detailliert ist, andererseits die Erfindung nicht genau beschreibt.

66 S. o. unter II.2.d).

67 Vgl. Gehring, Berliner Ansatz zu „Open Software Patents", http://ig.cs.tu-berlin.de/ap/rg/2000-05/Gehring-OpenSoftwarePatents-052000.pdf.

68 Gehring/Lutterbeck, in: Taeger/Wiebe (Hrsg.), S. 301, 316.

V. Fazit

Die genauere Betrachtung hat ergeben, dass es durchaus Konfliktpotential zwischen Patentschutz und Softwareentwicklung gibt. Andererseits besteht eine entsprechende Konfliktlage auch für andere patentierte technische Entwicklungen. Insbesondere die nähere Untersuchung der Open-Source-Software hat aufgezeigt, dass es durchaus bereits nach der bestehenden Rechtslage Möglichkeiten gibt, ohne Beeinträchtigung durch bestehenden Patentschutz Softwareentwicklung zu betreiben. Weitere Probleme sind durch mangelnde Erfahrung und einen noch nicht ausreichend dokumentierten Stand der Technik in diesem Bereich bedingt. Diese werden sich mit der Zeit abschwächen, wozu durch Dokumentation und Einrichtung entsprechender Stellen beigetragen werden kann.[69] Schließlich ist das Dual Licensing Indiz dafür, dass ein gewisser Wettbewerb möglich erscheint.[70]

Die größte Gefahr scheint von einer zu breiten Patentierung unter Einschluss von Algorithmen auszugehen. Gerade insoweit sind die bestehenden Grundsätze der Patentierung besonders zu beachten. Insbesondere ist die Rechtsprechung in Zukunft gefordert, noch stärker die Grenzlinie zwischen schutzfreien Algorithmen und geschützten innovativen Programmfunktionen herauszuarbeiten. Auch sind die hohen Anforderungen an Neuheit und erfinderische Tätigkeit strikt zu beachten. Unter diesen Voraussetzungen steht zu erwarten, dass die Patentierung computerimplementierter Erfindungen in gleichem Masse eine innovationsfördernde Wirkung haben können wie herkömmliche technische Erfindungen. Insoweit gelten die eingangs angeführten Vorteile der Patentierung. Das gilt gerade auch für den Markteinstieg von KMU, für den verschiedene Studien die positiven Effekte der Patenterteilung hervorgehoben haben.[71] Es bleibt zu hoffen, dass wir durch die Entschließung des Europäischen Parlaments zur Harmonisierung im Bereich computerimplementierter Erfindungen nicht in den früheren Zustand der Rechtsunsicherheit zurück geworfen werden.

[69] Vgl. auch die einerseits eher zuversichtliche Schlußfolgerung von Jaeger/Metzger, S. 127 f., andererseits die skeptische von Gehring/Lutterbeck, in Taeger/Wiebe (Hrsg.), S. 301, 321

[70] Vgl. auch Haase, S. 175.

[71] Vgl. Haase, S. 138 ff., m.w.Nachw.

Urheber- und Lizenzrecht im Bereich von Open-Source-Software

Olaf Koglin und Axel Metzger

1. Einführung in das Urheberrecht

Das Urheberrecht schützt „persönliche geistige Schöpfungen“ (§2 Abs. 2 Urheberrechtsgesetz – UrhG). Hierunter fallen unter anderem Gedichte und andere Sprachwerke, Musik, Filme, wissenschaftliche Zeichnungen – und Computerprogramme. Aus dem Erfordernis der persönlichen geistigen Schöpfung ergibt sich, dass das zu schützende Werk von einem Mensch geschaffen sein muss und eine *Schöpfungshöhe* aufweist, also nicht gänzlich trivial ist. Sind diese Voraussetzungen gegeben, entsteht das Urheberrecht und mit ihm der urheberrechtliche Schutz mit der Schöpfung des Werkes. Ein formeller Akt wie das Anbringen eines Copyright- oder Urheberrechtsvermerkes ist nicht erforderlich, aber auch nicht schädlich.

Das Urheberrecht schützt den Urheber sowohl in seinen geistigen und persönlichen Beziehungen zum Werk (sog. *Urheberpersönlichkeitsrecht*) als auch hinsichtlich der Nutzung des Werkes (sog. *Verwertungsrecht*). Das Urheberpersönlichkeitsrecht, das die Anerkennung der Urheberschaft und Schutz vor Entstellung des Werkes gewährt, spielt bei Computerprogrammen nur eine untergeordnete Rolle, kann aber im Bereich des Open-Content erheblich sein.

Wirtschaftlich steht das Verwertungsrecht im Vordergrund, das dem Urheber das Recht zur Verwertung seines Werkes verleiht. Grundsätzlich darf ein urheberrechtlich geschütztes Werk – im Rahmen der vom UrhG geschützten Verwertungsarten – ausschließlich durch den Urheber verwendet werden. Er kann jedoch anderen – gegebenenfalls gegen eine Gegenleistung – bestimmte *Nutzungen* gestatten.

Von diesem Ausschließlichkeitsrecht gibt es Ausnahmen: Zum einen sind im privaten oder öffentlichen Interesse bestimmte Nutzungen auch ohne besondere Erlaubnis des Urhebers gestattet (sog. Schranken des Urheberrechts, §§45 ff. UrhG). Hierunter fällt neben Nutzungen zu Bildungs- und Informationszwecken auch die sog. Privatkopie (§53 UrhG). Zum anderen sind bestimmte Nutzungen gar nicht erst urheberrechtlich geschützt. Das Lesen eines Buches zum Beispiel ist keine dem Urheber vorbehaltene Verwertungshandlung. Daher darf jedermann, ohne einer besonderen „Leselizenz“ zu bedürfen, ein urheberrechtlich geschütztes Buch lesen – selbst wenn das Buch urheberrechtswidrig hergestellt worden sein sollte (Koglin 2004, Kap. 4 A).

Das UrhG benennt katalogartig die dem Urheber vorbehaltenen Verwertungsrechte und trennt dabei zwischen der Verwertung des Werkes in körperlicher Form und der unkörperlichen Verwertung (§15 Abs. 1 / 2 UrhG). Zur körperlichen Verwertung gehört insbesondere die Vervielfältigung und die Verbreitung des Werkes einschließlich der Vermietung. Zur unkörperlichen Verwertung gehört insbesondere

die öffentliche Aufführung und Sendung eines Werkes. Das öffentliche Anbieten auf einem Server („making available to the public") als Angebot zur unkörperlichen Verbreitung scheint in dieses Raster nicht zu passen, weshalb bis vor kurzem umstritten war, wie es vom Urheberrecht erfasst wird. Der Gesetzgeber hat diese Lücke im Herbst 2003 mit dem „Gesetz zur Regelung des Urhebers in der Informationsgesellschaft" geschlossen und in das UrhG die neuen §19a und §69c Nr. 4 eingefügt. Hiernach ist das Recht der öffentlichen Zugänglichmachung das Recht, das Werk drahtgebunden oder drahtlos der Öffentlichkeit in einer Weise zugänglich zu machen, dass es Mitgliedern der Öffentlichkeit von Orten und zu Zeiten ihrer Wahl zugänglich ist. Dieses Recht ist gem. §15 Abs. 2 Nr. 2 UrhG als weiteres Verwertungsrecht dem Urheber vorbehalten.

1.1 Urheberrechte an Computerprogrammen

Computerprogramme gelten nach §2 Abs. 1 Nr. 1 UrhG ausdrücklich als Sprachwerke und sind somit urheberrechtlich schutzfähige Werke.

Die Entscheidung, Computerprogramme gerade durch das Urheberrecht zu schützen, ist weder zwingend noch unumstritten[1]. Als technische Problemlösung, die in aller Regel mehr an Funktionalität als an künstlerischer Individualität und Kreativität orientiert ist, wäre statt des urheberrechtlichen Schutzes auch die Protektion durch ein gewerbliches Schutzrecht in Betracht gekommen. Der Weg eines Sonderrechtsschutzes wurde vor allem deswegen abgelehnt, weil mit dem Instrument des Urheberrechts auf ein funktionierendes System internationaler Konventionen zurück gegriffen werden konnte. Außerdem hängt der urheberrechtliche Schutz nicht davon ab, ob ein Werk besonders künstlerisch gestaltet wurde, das Urheberrecht schützt zum Beispiel auch wissenschaftliche Zeichnungen oder Stadtpläne.

Allerdings stellte der BGH 1985 in der Entscheidung „Inkasso-Programm" (BGHZ 94, 276) noch sehr große Ansprüche an die Schöpfungshöhe von Computerprogrammen. 1993 wurde in Deutschland die EG-Richtlinie 91/250/EWG umgesetzt und das Urheberrecht an Computerprogrammen inhaltlich besonders geregelt. Unter der Überschrift „Besonderer Schutz für Computerprogramme" wurden die §§69a bis 69g in das UrhG aufgenommen. §69a Abs. 3 UrhG besagt, dass zur Bestimmung der Schutzfähigkeit von Computerprogrammen keine qualitativen oder ästhetischen Kriterien anzuwenden sind. Seitdem werden an die Schöpfungshöhe keine besonderen Anforderungen mehr gestellt, sodass jedes praktisch relevante Programm urheberrechtlich geschützt ist (eine Ausnahme machen hiervon allenfalls triviale „Hallo Welt"-Programme).

Zu beachten ist jedoch, dass die §§69a ff. UrhG nur für Computerprogramme gelten, welche vom weiteren Begriff der Software zu unterscheiden sind: Während Software als Gegenbegriff zur Hardware sämtliche digitalisierten Daten umfasst, ist ein Programm grundsätzlich ausführbar.[2] Digitalisierte Bilder oder Musikstücke unterfallen also nicht den besonderen Bestimmungen der §§69a ff. UrhG.

---

1 Zur Kritik siehe z. B. Hoeren in Möhring/Nicolini 2000, Vor §§69a ff.

2 Allerdings bestimmt §69a UrhG, daß auch der Entwurf und sämtliche Ausdrucksformen des Programms geschützt werden. Das Programm kann also auch fehlerhaft und insoweit nicht ausführbar sein. Auch kann ein Programm auf Papier ausgedruckt sein muss also nicht als Software vorliegen.

1.2 Besondere urheberrechtliche Bestimmungen für Computerprogramme

Computerprogramme unterliegen gem. §§69a bis 69g UrhG besonderen Bestimmungen. Neben der dargestellten Herabsenkung der Anforderungen an die Schöpfungshöhe durch §69a Abs. 3 UrhG wird bei Programmen, die in Arbeitsverhältnissen geschaffenen werden, das Verwertungsrecht durch §69b UrhG auf den Arbeitgeber verlagert.

Darüber hinaus ist bei Computerprogrammen die Benutzung durch Endnutzer stark eingeschränkt. Während bei einem „normalen" Werk wie einem Buch das Lesen oder Kaufen desselben keine urheberrechtlich geschützte Handlung ist, ist bei Computerprogrammen durch den besonderen Schutz des §69c UrhG auch das Ausführen eines Programms eine dem Urheber vorbehaltene Verwertungshandlung. Eine Ausnahme hiervon macht jedoch §69d Abs. 1 UrhG, wonach mangels abweichender Vereinbarung das Ausführen, Vervielfältigen und Bearbeiten des Programms zur bestimmungsgemäßen Benutzung des Programms einschließlich der Fehlerberichtigung jedem „zur Verwendung eines Vervielfältigungsstücks des Programms Berechtigten" gestattet ist. Dies gewährleistet, dass der Erwerber eines Vervielfältigungsstücks das Programm auch benutzen, d.h. ausführen darf.

Zur Verwendung des Programms berechtigt ist nicht allein der Käufer oder Lizenznehmer persönlich, sondern auch Angestellte und Familienmitglieder. Gleichwohl stellen §§69c, 69d UrhG gegenüber sonstigen Werken eine erhebliche Erweiterung der dem Urheber vorbehaltenen Verwertungen oder, andersherum betrachtet, eine starke Einschränkung der Rechte der Allgemeinheit dar. Auch ist unklar, wie §69d UrhG beim Download von Computerprogrammen anzuwenden ist (Koglin 2004, Kap. 4A).

Zusätzlich zu dieser Ausweitung des Schutzes auf Handlungen, die die bestimmungsgemäße Benutzung des Werkes durch Endkunden darstellen, sind bei Computerprogrammen auch die Schranken des Urheberrechts zu Lasten der Verbraucher eingeengt. Insbesondere dürfen lediglich einzelne Sicherungskopien, aber keine darüber hinausgehenden Privatkopien erstellt werden.

1.3 Einräumung von Nutzungsrechten

Nach der Schöpfung eines Werkes ist also fremden Personen die Nutzung des Werkes auf eine der dem Urheber vorbehaltenen Nutzungsarten zunächst untersagt. Um anderen die Nutzung zu gestatten, kann der Urheber Rechte zur Nutzung seines Werkes einräumen.

Diese sogenannten „Nutzungsrechte" können gemäß §31 Abs. 1 S. 2 UrhG als einfaches oder als ausschließliches Recht eingeräumt werden. Das einfache Nutzungsrecht berechtigt gemäß §31 Abs. 2 UrhG den Nutzungsrechtsinhaber, das Werk neben dem Urheber oder anderen Berechtigten auf die vereinbarte Art zu nutzen. Das ausschließliche Nutzungsrecht hingegen berechtigt gem. §31 Abs. 3 UrhG allein (also ausschließlich) dessen Inhaber zur vereinbarten Nutzung. Dabei sind nicht nur Dritte, sondern auch der Urheber selbst von dieser Nutzung ausgeschlossen.

Nutzungsrechte können gem. §31 Abs. 1 UrhG zeitlich, räumlich und auf bestimmte *Nutzungsarten* beschränkt werden. Die Nutzungsarten sind dabei nicht deckungsgleich mit den oben dargestellten Verwertungsrechten (§§15 ff. UrhG), sondern können in einem gewissen Rahmen vom Urheber frei zugeschnitten werden.

So kann zum Beispiel das Nutzungsrecht zur Verwertung eines Romans auf die Herstellung und Verbreitung von Taschenbüchern und gleichzeitig auf eine bestimmte Auflage und auf das Gebiet der Bundesrepublik Deutschland beschränkt werden.

Bei Überschreiten einer solchen Beschränkung agiert der Inhaber des beschränkten Nutzungsrechts außerhalb des Nutzungsrechts und damit urheberrechtswidrig. Da er sich dadurch strafbar machen kann, werden an die Erkennbarkeit der Beschränkung des Nutzungsrechts bestimmte Anforderungen gestellt. Auch sind preisliche Vorgaben – insbesondere das Verlangen eines Mindestpreises beim Verkauf des Werkes – nicht mit urheberrechtlicher Wirkung möglich (Koglin 2004, Kap. 4 C). Detaillierte Beschränkungen der Nutzung sind jedoch durch den Abschluss eines entsprechenden Vertrags möglich (dazu näher in Abschnitt 2.0).

1.4 Zweckübertragungslehre und Erschöpfungsgrundsatz

Mit dem *Zweckübertragungsgrundsatz* normiert §31 Abs. 5 UrhG ein fundamentales Prinzip bei der Einräumung von urheberrechtlichen Nutzungsrechten. Hiernach räumt der Urheber Nutzungsrechte im Zweifel nur soweit ein, wie es nach dem Zweck der Rechtseinräumung erforderlich ist Schricker §37 Rn. 4).

Der *Erschöpfungsgrundsatz* ergänzt zu den Einschränkungen bei der inhaltlichen Beschränkbarkeit von Nutzungsrechten die Interessen der Allgemeinheit an einem freien Warenverkehr: Sind Kopien des Werkes mit Zustimmung des zur Verbreitung berechtigten in der EU in den Verkehr gekommen, so ist die weitere Verbreitung auch ohne Erlaubnis des Urhebers zulässig – das Verbotsrecht des Urhebers ist dann insoweit „erschöpft".

Hiervon ausgenommen ist jedoch die Vermietung des Werkes. Wer also ein Buch oder eine Original-CD gekauft hat, darf diese auf dem Flohmarkt oder bei eBay zu jedem erzielbaren Preis verkaufen. Dies gilt sowohl bei Computerprogrammen als auch bei sonstigen Werken (§17 Abs. 2, 69c Nr. 3 S. 2 UrhG).

1.5 Die „Linux-Klausel" (§ 31 Abs. 3 S. 3 UrhG)

Für die Einräumung von Nutzungsrechten kann der Urheber die Gewährung einer Gegenleistung – insbesondere die Zahlung eines einmaligen oder regelmäßigen Geldbetrags – verlangen. Theoretisch kann der Urheber deren Höhe frei aushandeln. In vielen Marktsegmenten stehen Autoren und Künstler den Verwertungsunternehmen jedoch nicht wirtschaftlich gleichwertig gegenüber.

Mit einer am 1. 7. 2002 in Kraft getretenen Urheberrechtsnovelle bezweckte der Gesetzgeber, Urhebern und ausübenden Künstlern einen Anspruch auf eine angemessene Vergütung zu geben, der nicht abbedungen werden kann (§32 UrhG). Falls ein Werk später wesentlich populärer wird als bei Einräumung des Nutzungsrechts angenommen, oder falls die Vergütung des Urhebers aus anderen Gründen nicht

angemessen ist, kann der Urheber somit die Differenz zur angemessenen Vergütung nachfordern.

Dies hätte jedoch in Deutschland das System Freier Software in Frage gestellt: Personen, die zum Beispiel Anfang der 90er Jahre an Linux mitgearbeitet und daran Urheberrechte erworben haben, könnten Jahre später von Distributoren und anderen Nutzern eine „angemessene Vergütung" verlangen.

Auf Initiative des Instituts für Rechtsfragen der Freien und Open-Source-Software (vgl. ifrOSS[3]) wurde in §31 Abs. 3 S. 3 UrhG die „Linux-Klausel" aufgenommen. Hiernach kann der Urheber auf die angemessene Vergütung verzichten, wenn er unentgeltlich ein einfaches Nutzungsrecht an jedermann einräumt.

1.6 Internationales Urheberrecht

Da jeder Staat in seinem eigenem Territorium eine eigene Gesetzgebungsbefugnis hat, kann ein Parlament nicht über die Rechtslage in einem anderen Staat bestimmen. Im Urheber- und sonstigen Immaterialgüterrecht kann daher jeder Staat nur regeln, welche Werke in seinem Territorium auf welche Weise und in welchem Umfang geschützt werden (sog. „Territorialitätsprinzip").

Durch internationale Verträge wie der Berner Übereinkunft und später der Revidierten Berner Übereinkunft (RBÜ), wurden bereits im 19. Jahrhundert Mindestanforderungen und Grundsätze des urheberrechtlichen Schutzes und des Schutzes der Werke von Ausländern vereinbart, die durch die jeweiligen Urheberrechtsgesetze der Mitgliedsstaaten in nationales Recht umgesetzt wurden (Argument damals wie heute: Immaterielle Güter können sonst allzu schnell kopiert und über Grenzen hinweg verbreitet werden). Somit existieren weltweit in den meisten Staaten ähnliche Urheberrechtsgesetze, die jedoch im Detail variieren können. So schreibt Art. 7 Abs. 1 RBÜ nur vor, dass das Urheberrecht mindestens bis 50 Jahre nach dem Tod des Autors bestehen muss. In Deutschland und vielen anderen Staaten erlischt das Urheberrecht aber erst 70 nach dem Tod des Urhebers (§64 UrhG).

Konventionen wie die RBÜ schützen jedoch nicht unmittelbar ein Werk, sondern verpflichten die Vertragsstaaten nur zur Umsetzung der in der Konvention niedergelegten Regelungen in nationale Gesetze. Der oft gelesene Satz „This Program is protected by U.S. Copyright Law and by international treaties" ist also falsch: Nach dem Territorialitätsprinzip wird ein Programm in den USA durch den dortigen „Copyright Act" geschützt, in Deutschland allein durch das deutsche UrhG und in Frankreich allein durch den „Code de la Propriété Intellectuelle".

2. Vertragsrecht

Zusätzlich zu der Einräumung von Nutzungsrechten kann der Urheber bzw. ein Inhaber von Verwertungsrechten mit dem Nutzer einen Vertrag schließen, in dem die Einzelheiten der Nutzung und sonstige Verpflichtungen geregelt werden. Hierin können zum Beispiel Beschränkungen auferlegt werden, die mit urheberrechtlicher Wirkung nicht möglich sind.

[3] Vgl. http://www.ifrOSS.de.

Allerdings greift im Vertragsrecht der Schutz des Rechts der Allgemeinen Geschäftsbedingungen, das Rechtsgebiet, das Kunden vor dem „Kleingedruckten" schützen soll. Nach dem sog. *AGB-Recht* ist zum Beispiel ein genereller Haftungs- und Gewährleistungsausschluss unzulässig. Die verwendeten Klauseln müssen verständlich sein, anderenfalls sind sie nicht Bestandteil des Vertrags; Unklarheiten gehen zu Lasten der Vertragspartei, von der der Formulartext stammt (sog. „*Unklarheitenregel*"[4]).

3. Lizenzrecht

In Lizenzvereinbarungen werden typischerweise die Verpflichtung des Urhebers bzw. Rechteinhabers zur Einräumung von Nutzungsrechten, die Erfüllung dieser Pflicht sowie weitere vertragliche Vereinbarungen geregelt. Diese Regelungen müssen innerhalb der Lizenz nicht zwingend voneinander getrennt werden. Gleichwohl enthalten sie sowohl urheber- als auch vertragsrechtliche Komponenten, weshalb auch der Begriff „*Lizenzvertrag*" verwendet wird. Dieser Begriff trennt zudem zwischen dem sog. Verpflichtungsgeschäft, mit dem die Parteien vertrags- und urheberrechtliche Rechte bzw. Pflichten vereinbaren, und dem zu erteilenden Nutzungsrecht selbst, das ebenfalls „Lizenz" genannt wird. Gerade bei Open-Source-Software ist der Begriff „Softwareüberlassungsvertrag" irreführend, da durch Open-Source-Lizenzen nicht unmittelbar der Verkauf oder die Überlassung der Software, sondern nur die Einräumung von Rechten daran geregelt wird.

Lizenzverträge über die Nutzung von Programmen durch Endnutzer („End User License Agreements", EULAs) stammen regelmäßig vom Urheber bzw. Rechteinhaber. Damit schlagen in Softwarelizenzverträgen regelmäßig zwei Herzen: Der Kern des Lizenzvertrags, die Einräumung von Nutzungsrechten, unterliegt dem Zweckübertragungsgrundsatz und erfolgt damit in Zweifelsfällen zu Gunsten des Lizenzgebers in möglichst kleinem Umfang. Das „Kleingedruckte" drumherum wird hingegen mit der AGB-rechtlichen Unklarheitenregel in Zweifelsfällen zu Lasten des Lizenzgebers ausgelegt.

Softwarelizenzverträge und die mit ihnen bezweckte Einräumung von Nutzungsrechten bedürfen keiner besonderen Form, insbesondere nicht der Schriftform. Lediglich Verträge, mit denen sich der Urheber zur Einräumung an zukünftigen Werken verpflichtet, die noch nicht näher bestimmt sind, müssen gem. §40 UrhG schriftlich abgeschlossen werden.

4. Open Source-Lizenzen

Was passiert nun urheberrechtlich bei der Lizenzierung eines Programms unter einer Open-Source-Lizenz?

Wie auch sonst im deutschen Urheberrecht ist ein Verzicht auf das Urheberrecht nicht möglich. Wie auch immer ein entsprechender Lizenzvertrag gestaltet sein mag, kann er wegen des Grundsatzes der Unübertragbarkeit des Urheberrechts gem. §29 Abs. 1 UrhG nicht zu einer völligen Ablösung des Rechts vom Urheber

4 Vgl. dazu Koglin (2004, Kap. 5).

führen. Die Lizenzierung eines Programms als „Freie Software" bedeutet also nicht, dass sich die Urheber oder sonstigen Rechtsinhaber außerhalb aller Rechte begeben.

Wer die GNU-GPL oder die anderen Lizenzen durchliest, wird wohl zustimmen, dass es den Lizenzgebern auch gar nicht um eine völlige Preisgabe der Rechte geht. Vielmehr werden Nutzungsrechte eingeräumt, zugleich werden den Lizenznehmern bestimmte Verpflichtungen auferlegt. Wie wollte man diese konstruieren, wenn die Lizenzgeber ganz auf ihre Verbotsrechte verzichtet hätten? Kurzum: Ein vollständiger Verzicht ist weder gesetzlich möglich noch nach der Gestaltung der Lizenzen gewünscht.

Welche Nutzungsrechte werden dem Lizenznehmer eingeräumt? Im Folgenden soll die „GNU General Public License" (GPL), die am weitesten verbreitete Open-Source-Lizenz und lizenzrechtliche Grundlage der Entwicklung und des Vertriebs des Betriebssystems GNU/Linux, als Beispiel näher betrachtet werden.

4.1 Vervielfältigen und Verbreiten (GPL)

Nach Ziffer 1 der GPL wird jedem Lizenznehmer die Vervielfältigung und Verbreitung unveränderter Versionen des Programms gestattet: „You may copy and distribute verbatim copies of the Program".

Hier fangen die Probleme schon an. Eindeutig gestattet wird zunächst also die Herstellung körperlicher Vervielfältigungsstücke im Sinne von §69c Nr. 1 UrhG. Jeder Lizenznehmer erhält ein einfaches Nutzungsrecht, Kopien anzufertigen. Genauso klar ist die Verbreitung körperlicher Vervielfältigungsstücke im Sinne von §69c Nr. 2 UrhG.

Der Begriff der „Verbreitung" umfasst nach deutscher Urheberrechtsterminologie aber nur die Verbreitung körperlicher Kopien. Für die unkörperliche Verbreitung, sei es das Angebot zum Download, sei ein bloßes „Streaming", also eine Wiedergabe des Programms, welche beim Nutzer gebraucht werden kann, ohne dass es zur Herstellung einer Kopie auf seinem Prozessor kommt, ist im deutschen Recht ein eigenes Verwertungsrecht vorgesehen – das Recht der öffentlichen Zugänglichmachung gem. §69c Nr. 4 UrhG.

Geht man für die Auslegung der GPL von einer schlichten Übersetzung des Begriffs „distribute" aus und richtet sich ansonsten nach der Terminologie des Urheberrechtsgesetzes, so wären die genannten Formen der unkörperlichen Verbreitung nicht gestattet (Koch 2000b: 338). Hierfür scheint auch die Zweckübertragungslehre aus §31 Abs. 5 UrhG zu sprechen, nach welcher Nutzungsrechtseinräumungen im Zweifel eng auszulegen sind. Es erscheint indessen überzeugender, den Begriff „distribute" weit auszulegen. Nach der us-amerikanischen Terminologie sind von dem Begriff auch unkörperliche Nutzungsarten umfasst.[5] Auch wenn für das deutsche Hoheitsgebiet deutsches Urheberrecht zur Anwendung zu bringen ist, bedeutet dies nicht, dass die Lizenz völlig herausgelöst aus ihrer internationalen Zielrichtung interpretiert werden muss.

[5] Vgl. §106 (3) U.S. Copyright Act.

4.2 Veränderung des Programms (GPL)

Ziffer 2 der GPL bringt eine weitere zentrale Berechtigung der Lizenznehmer. Diesen ist die Veränderung des Programms gestattet sowie die Vervielfältigung und Verbreitung entsprechend veränderter Programmversionen.

Diese Rechtseinräumung wirft ebenfalls gewisse Probleme auf. Nach deutschem Urheberrecht steht jedem Urheber als Ausprägung des Urheberpersönlichkeitsrechts die Befugnis zu, sich gegen solche Veränderungen seines Werks zur Wehr zu setzen, die seine persönlichen und geistigen Interessen beeinträchtigen. Auf dieses Recht kann nicht im Vorweg durch eine pauschale Erlaubnis verzichtet werden.

Für Software kommt eine Gefährdung der persönlichen Interessen allerdings nur in Extremfällen in Betracht. Es ist zwar zu beachten, dass gerade die Urheber von Open-Source-Software oftmals ideelle Interessen mit ihren Programmen verbinden (häufig wird die Ähnlichkeit zur künstlerischen Tätigkeit hervorgehoben). Es gilt aber andererseits auch zu berücksichtigen, dass die Veränderungsfreiheit elementarer Bestandteil der Philosophie der Freien Software ist. Man wird deswegen berücksichtigen müssen, dass sich die Urheber ohne jeden wirtschaftlichen oder organisatorischen Druck zum Gebrauch entsprechender Lizenzen entschließen. Entsprechende Ansprüche sind deswegen nicht unmöglich, aber doch eher unwahrscheinlich (Schulz 2004 Kap. 1C 3b).

4.3 Grundpflichten der Lizenznehmer

Alle gängigen Open-Source-Lizenzen verknüpfen die Einräumung von Nutzungsrechten mit einem Katalog von Pflichten, die den Lizenznehmern auferlegt werden. Man kann dabei zwischen „Grundpflichten“, die sich in allen Lizenzen nahezu identisch finden, und „Copyleft“-Klauseln unterscheiden, die lediglich in einigen Lizenzen vorgesehen sind.

Die Grundpflichten sorgen für die Funktionsfähigkeit des Vertriebs der Programme. Alle Lizenzen sehen zum einen vor, dass mit jedem Vervielfältigungsstück der Software stets auch eine Kopie der maßgeblichen Lizenz verbreitet werden muss. Da es keine zentralen Vertriebsstrukturen für Open-Source-Software gibt, kann nur auf diese Weise sicher gestellt werden, dass tatsächlich jeder neue Nutzer eines Programms erfährt, welche Rechte er an der Software erwerben kann.

Diesem Zweck dient auch die Verpflichtung der Nutzer, Hinweise in dem Programm, welche auf die Geltung der Lizenz verweisen, nicht zu verändern. Nur wenn in oder auf dem Programm vermerkt ist, welche Lizenz für das jeweilige Programm maßgeblich ist, kann eine Verbindung zwischen Software und Lizenz gezogen werden.

Die Grundpflichten stellen die meisten Lizenznehmer vor keine oder nur geringe Probleme. Es kann allenfalls für Unternehmen, die beispielsweise in Massenproduktion hergestellte Handys mit GNU/Linux ausstatten, Kosten verursachen, die Bedienungsanleitung um die entsprechende Seitenzahl für den Lizenztext auszuweiten.

4.4 „Copyleft"-Klauseln

Im Mittelpunkt des Interesses stehen die so genannten „Copyleft"-Klauseln. Ziffer 2b der GPL ist der Urtypus entsprechender Lizenzklauseln. Zahlreiche andere Lizenzen sehen entsprechende Lizenzbestimmungen vor, wenngleich in den Detailfragen zahlreiche Unterschiede bestehen.

„Copyleft"-Klauseln verpflichten den Lizenznehmer, der bei der Veränderung des Programms selbst Urheberrechte erwirbt, diese entsprechend den Bedingungen der Lizenz anderen zur Verfügung zu stellen. Die Verpflichtung zur Freigabe von Modifikationen greift allerdings nur für den Fall, dass sich der Lizenznehmer dazu entschließt, seine veränderte Version zu verbreiten. Es wird also niemand ans Licht der Öffentlichkeit gezwungen.

Entsprechende Lizenzbestimmungen werfen gerade für Unternehmen eine Reihe schwieriger Abgrenzungsfragen auf. Im Mittelpunkt steht dabei die folgende Fragen: In welchem Fall stellt das Hinzufügen von Code eine Veränderung des vorbestehenden Programms dar, sodass die „Copyleft"-Klausel greift? Wann liegt eine bloße Zusammenstellung zweier unabhängiger Programme vor, welche keine Verpflichtungen nach sich zieht?

Die Lizenzen sind hier unterschiedlich klar gefasst. Während die GPL in Ziffer 2 mehrere, nicht sehr deutliche Kriterien nebeneinander stellt, entscheidet nach Ziffer 1.9 der „Mozilla Public License", der für den Webbrowser Mozilla maßgeblichen Lizenz, alleine die Technik der Zusammenstellung. Wird hinzugefügter Code in einem selbständigen File abgespeichert, so liegt keine Modifizierung des ursprünglichen Programms vor.

Andere Lizenzen, insbesondere die „BSD"-Lizenz und die „Apache Software License," sehen keine „Copyleft"-Klausel vor. Dem Nutzer steht es dadurch frei, Fortentwicklungen in einem herkömmlichen Lizenzmodell zu verbreiten.

5. Rechtliche Bindungswirkung

Trotz aller Unterschiede in den juristischen Einzelfragen, werden die Lizenzen einschließlich der Verpflichtungen der Lizenznehmer in den meisten Fällen als rechtlich bindend eingestuft. Voraussetzung hierfür ist nach deutschem Recht der Abschluss eines wirksamen Vertrags zwischen den Lizenzgebern und dem Lizenznehmern. Dabei gilt es zwei Vertragsgegenstände zu unterscheiden: die Software als solche und die soeben beschriebenen urheberrechtlichen Nutzungsrechte.

Wer eine klassische Linux-Distribution von SuSE, Redhat, Debian usw. in einem Kaufhaus erwirbt, schließt zunächst nur mit dem Kaufhaus einen Vertrag über die Überlassung der Software. Dieser Vertrag wird überwiegend als Kaufvertrag eingestuft (Jaeger und Metzger 2002 S. 165). Der normale Softwarenutzer erhält dadurch zugleich die gesetzlichen Mindestrechte aus §§69d, 69e UrhG, welche ihm die Benutzung, die Fehlerbeseitigung sowie die Erstellung einer Sicherungskopien gestatten. Wer also ein Open-Source-Programm lediglich benutzt, ohne die besonderen Befugnisse aus den Lizenzen wahrzunehmen, kommt mit den Lizenzen nicht in Kontakt.

Einige juristische Autoren gehen demgegenüber zu Unrecht davon aus, dass die Open-Source-Lizenzen in gleicher Weise wie sonstige Schutzhüllenverträge oder „Clickwrap"-Verträge unwirksam seien, weil dem Erwerber keine zumutbare Möglichkeit der Kenntnisnahme während des Erwerbs der Software gegeben wird (Omsels 2000 S. 151 ff.) Diese Ansicht verkennt aber, dass die Rechte aus den Lizenzen für die normale Benutzung des Programms unerheblich sind. Teilweise erklären die Lizenzen ausdrücklich, dass das bloße Laufenlassen außerhalb ihres Anwendungsbereichs liegt.[6]

Es kommt deswegen in aller Regel erst dann zum Abschluss des oben beschriebenen Lizenzvertrags, wenn der Nutzer das Programm vervielfältigen, verbreiten oder über das gesetzlich gestattete Maß hinaus verändern möchte. Die Lizenztexte stellen sich aus vertragsrechtlicher Sicht als Angebot an jedermann auf Abschluss eines Lizenzvertrags dar. Der Nutzer nimmt dieses Angebot durch schlüssiges Verhalten an, wenn er die Befugnisse aus den Lizenzen in Anspruch nimmt. Diese Annahmeerklärung muss nach deutschem Vertragsrecht den Rechtsinhabern nicht zugehen (Koglin 2004 Kap 5A).

Besondere vertragsrechtliche Schwierigkeiten ergeben sich für den Fall, dass der Lizenznehmer ein Verbraucher ist. Nach europäischen Verbraucherschutzrecht sind fremdsprachliche Standardverträge dem Verbraucher nur ausnahmsweise zuzumuten. Einige Autoren gehen deswegen davon aus, dass die in englischer Sprache verfassten Open-Source-Lizenzen gegenüber Verbrauchern nicht wirksam sind (Spindler 2004, 68).

Letztlich erscheint die Sprachproblematik aber nicht als unlösbares Problem. Es ist im deutschen Vertragsrecht anerkannt, dass es auch für einen Verbraucher nicht möglich ist, einerseits die Rechte aus der Lizenz zu erwerben, also die für ihn positive Seite anzunehmen, und sich andererseits im Hinblick auf die Verpflichtungen darauf zu berufen, dass er diese nicht habe verstehen können. Ein solches Verhalten wäre rechtsmissbräuchlich.

Im Übrigen ist darauf hinzuweisen, dass Konfliktfälle bei Verbrauchern ohnehin eher theoretischer Natur sind. Welcher Urheber eines Open-Source-Programms würde es auf sich nehmen, einen Verbraucher zu verklagen, weil dieser seinem Nachbarn eine Kopie seines Programms gegeben hat, ohne auch die Lizenz mitzuliefern? Streitfälle werden in aller Regel gewerbliche Lizenznehmer betreffen und diesen gegenüber macht die vertragliche Einbeziehung der Lizenzen keine Probleme. Es wäre gleichwohl wünschenswert, dass die Open-Source-Community die Herausforderung annimmt, ihre Lizenzbedingungen auf den internationalen Vertrieb anzupassen.

Gewisse Probleme bereitet schließlich der bereits erwähnte Erschöpfungsgrundsatz. Ist ein Vervielfältigungsstück einmal rechtmäßig in den Verkehr gebracht worden, so kann der Rechtsinhaber einen weiteren Vertrieb nicht mehr verbieten. Dies bedeutet zugleich, dass es dem Rechtsinhaber nicht mehr möglich ist, den Vertrieb an besondere Bedingungen zu knüpfen.

[6] Vgl. zum Beispiel Ziffer 0 Abs. 2 der GPL.

Man stelle sich etwa den Großhändler vor, der einen großen Posten GNU/Linux-Distributionen kauft und dann ohne Beachtung der sich aus der Open-Source-Lizenz ergebenden Grundpflichten vertreibt, also insbesondere ohne Kopie der maßgeblichen Lizenz. Hier stehen den Rechtsinhabern keine rechtlichen Mittel zur Verfügung, einen weiteren Vertrieb zu untersagen (Spindler und Wiebe 2003, 877).

Freilich scheint auch dieses Problem eher theoretischer Natur zu sein. Zumeist werden die Vervielfältigungsstücke elektronische Kopien der Lizenzen enthalten, auch befindet sich regelmäßig der Hinweis auf die jeweilige Lizenz im Code selbst. Welche wirtschaftlichen Erwägungen sollten den Großhändler dazu treiben, nunmehr auf jeder CD die entsprechenden Daten zu löschen? Praktisch erscheinen die aus dem Erschöpfungsgrundsatz resultierenden Probleme also durchaus handhabbar.

Zusammenfassend kann gesagt werden, dass die Lizenzen nach deutschem Urheberrecht und Vertragsrecht in den meisten Fällen in vollem Umfang bindend sind.

6. Open Content

Das Lizenzmodell „Open-Source" wird seit wenigen Jahren auch verstärkt auf andere Werkgattungen, also Texte, Datenbanken, Grafiken usw. angewandt. Für diese Versuche scheint sich der Begriff „Open-Content" durchzusetzen. Aus deutscher Sicht hervorzuheben ist die vom Universitätsverband Nordrhein-Westfalen in Zusammenarbeit mit dem ifrOSS entwickelte „Lizenz für freie Inhalte"[7]. International ist das Projekt „Creative Commons"[8] am weitesten entwickelt.

Open-Content-Lizenzen stehen rechtlich vor den gleichen Problemen wie Open-Source-Lizenzen (Jaeger und Metzger 2003). Teilweise ergeben sich aber auch Unterschiede. So ist zum Beispiel wegen des stärker ausgeprägten Persönlichkeitsbezugs eine völlige Veränderungsfreiheit der Lizenznehmer rechtlich problematisch. Die „Lizenz für freie Inhalte" sieht deswegen in Ziffer 3b) einen ausdrücklichen Vorbehalt der persönlichkeitsrechtlichen Ansprüche des Urhebers vor.

Zu beachten sind auch die Besonderheiten bei der Nennung der Urheber. Autoren von Texten haben ein lebhaftes Interesse daran, nicht mehr als Urheber genannt zu werden, wenn die inhaltlichen Aussagen eines Textes verändert werden (Koglin 2003).

Das Potential für „Open-Content" scheint gegenwärtig noch nicht voll ausgeschöpft. Gerade die sog. „Wissenschafts-Communities" arbeiten oftmals nach „Open-Content"-ähnlichen Grundsätzen, ohne ihre Regeln entsprechend verfasst zu haben. Die Erarbeitung von Lizenzmodellen könnte hier zu einer Klärung und Konfliktvermeidung führen.

7 Vgl. http://www.uvm.nrw.de/Lizenzen/uvm-lizenz1html.htm.
8 Vgl. http://creativecommons.org.

7. Ausblick Urheberrecht

Die gegenwärtige europäische Rechtsentwicklung deutet auf eine weitere Verstärkung der geistigen Eigentumsrechte hin (Metzger/Wurmnest). Ein Vorschlag für eine Richtlinie über die „Durchsetzung geistiger Eigentumsrechte" befindet sich im Gesetzgebungsverfahren. Auf nationaler Ebene scheint sich eine Verlangsamung dieser expansiven Tendenz abzuzeichnen. Eine Fachgruppe des Bundesjustizministeriums verhandelt zurzeit über einen sog. „Zweiten Korb" für das Gesetz zum Urheberrecht in der Informationsgesellschaft.

Open Source wird von diesen Reformen nur mittelbar betroffen sein. Es bleibt gleichwohl zu hoffen, dass der Gesetzgeber auch künftig entsprechend dem Beispiel „Linux-Klausel" folgt und bereits im Gesetzgebungsverfahren dafür sorgt, dass sich die Bedingungen für Open Source jedenfalls nicht verschlechtern. Gleichzeitig sind die Entwicklergemeinschaften aufgefordert, ihre Lizenzmodelle weiter zu entwickeln und zu verbessern, um der oben angesprochenen Probleme Herr zu werden.

Literatur

ifrOSS: *Eingabe des Instituts für Rechtsfragen der Freien und Open Source Software*, online http://www.ifrOSS.de/ifrOSS_html/urhebervertragsrecht.pdf

Jaeger, Till / Metzger, Axel (2002): *Open Source Software: Rechtliche Rahmenbedingungen der Freien Software*, München, C. H. Beck

Jaeger, Till/Metzger, Axel (2003): *Open Content-Lizenzen nach deutschem Recht*, MMR (Zeitschrift für Informations-, Telekommunikations- und Medienrecht), S. 431

Koch: *Urheber- und Kartellrechtliche Aspekte der Nutzung von Open Source Software*, CR (Computer und Recht), S. 273 und 333

Koglin (2003): *Open Content-Lizenzen*, Linux-Magazin 10/2003, S. 70

Koglin (2004): *Opensourcerecht – Die urheber- und vertragsrechtlichen Beziehungen zwischen Lizenzgeber und Lizenznehmer bei Open Source Software am Beispiel der GPL*, Bonn

Metzger / Wurmnest (2003): *Auf dem Weg zu einem Europäischen Sanktionenrecht des geistigen Eigentums?*, ZUM (Zeitschrift für Urheber- und Medienrecht), S. 922.

Hoeren: in Möhring / Nicolini (2000): *Kommentar zum Urheberrechtsgesetz*, München

Omsels (2000): *Open Source und das deutsche Urheber- und Vertragsrecht (Beitrag zur Festschrift für Paul Hertin)*, München, S. 141

Schricker (1999): *Kommentar zum Urheberrechtsgesetz*, München

Schulz (2004): *Dezentrale Softwareentwicklungs- und Softwarevermarktungskonzepte – Vertragsstrukturen in Open Source Modellen*, Hamburg

Spindler (2004): *Rechtsfragen bei Open Source*

Spindler / Wiebe, Andreas (2003): *Open Soruce-Vertrieb, Rechteeinräumung und Nutzungsberechtigung*, CR (Computer und Recht), S. 873

Open Source Software – Ein Weg aus der Abhängigkeitsfalle zurück zur unternehmerischen Freiheit

UWE KÜSTER

1 Einleitung

Am 16. März 2002 fasste der Ältestenrat des Deutschen Bundestages den Entschluss, Linux als zukünftiges Betriebssystem auf ca. 200 Servern einzusetzen. Er folgte damit den Empfehlungen seiner IuK-Kommission. Die von der SPD-Fraktion initiierte und von Bündnis90/Die Grünen mitgetragene Empfehlung wurde teilweise als halbherzig und als lauer politischer Kompromiss bezeichnet. In Wirklichkeit aber ist sie richtungweisend und ein deutliches Signal für öffentliche Verwaltung und Industrie, verstärkt Open-Source-Software (OSS) einzusetzen.

Der Erfolg von OSS liegt gerade in ihrer Offenheit: Funktionsprinzipien und fundamentale Designentscheidung werden in Entwicklerteams öffentlich diskutiert. Die Implementation kann von jedem, der über das nötige Wissen verfügt, frei von Beschränkungen analysiert werden. Freie Betriebssysteme helfen aber nur zum Teil, die marktbeherrschende Stellung von Microsoft bei Office-Software aufzubrechen. Um hier nennenswerte Fortschritte zu erzielen, muss nicht nur quelloffene Software vorhanden sein, sondern die Idee offener Standards ist auch konsequent auf die Formate anzuwenden, in denen die Daten gespeichert werden.

Dieser Artikel will den Gang des Entscheidungsprozesses in der IuK-Kommission skizzieren. Ausgehend vom organisatorischen Aufbau des Bundestages in Abschnitt 2, werden in Abschnitt 3 die Migrationsvarianten und die Entscheidungskriterien dargelegt. Abschnitt 4 geht auf die politischen Aspekte und die strategischen Überlegungen ein. Abschnitt 5 wird einige der Folgen der Bundestagsentscheidung diskutieren. In Abschnitt 6 werden Projekte skizziert, die eng mit offenen Dokumentenstandards verknüpft sind.

2 Der Deutsche Bundestag

2.1 Organisatorischer Aufbau

Der Deutsche Bundestag ist kein homogen strukturiertes Gebilde. Derzeit gehören 603 Abgeordnete dem Parlament an. Die Abgeordneten einer Partei schließen sich zur besseren Koordinierung der Arbeit und zur Erzielung von Mehrheiten in Fraktionen zusammen. Jeder Abgeordnete hat in Berlin ein Büro, in dem er im Schnitt zwei Mitarbeiter beschäftigt. Diese Mitarbeiter sind dem Abgeordneten zugeordnet, nicht der jeweiligen Fraktion. Damit existieren im Abgeordnetenbereich in Berlin ca. 1800 Arbeitsplätze.

Im Wahlkreis hat jeder Abgeordnete mindestens ein weiteres Büro mit einem oder zwei Mitarbeitern. Diese rund 600 Arbeitsplätze sind über das ganze Land verteilt, und werden durch die Verwaltung des Bundestages betreut. Es ist hilfreich, sich die 603 Abgeordnetenbüros wie 603 unabhängige Firmen vorzustellen, die alle ihre eigenen Vorstellungen von Arbeitsorganisation haben – und auch umsetzen.

Die Fraktionen unterhalten zur Organisation und Unterstützung der politischen Facharbeit eigene Mitarbeiterstäbe, deren Größe sich an der Stärke der Fraktion orientiert. Derzeit beschäftigen alle Fraktionen zusammen knapp 800 Mitarbeiter.

Auch die Arbeit der Ausschüsse und des Plenums benötigt eine gute Vorbereitung. Das ist Aufgabe der Bundestagsverwaltung. In diesem Bereich sind knapp 2200 Menschen beschäftigt. In Summe finden sich im Bundestag etwa 5400 Arbeitsplätze, die mit PCs ausgestattet sind. Die eingesetzte Hard- und Software differiert auf Grund der unterschiedlichen Erfordernisse.

Der Ältestenrat ist das Organ des Bundestages, das den notwendigen Abstimmungs- und Koordinierungsbedarf deckt. Der Ältestenrat dient als gemeinsames Beratungsorgan zur Steuerung der Arbeit des Parlaments. Ihm gehören neben dem Präsidenten und den Vizepräsidenten 23 weitere Abgeordnete an. Diese werden von den Fraktionen entsprechend ihrer Mitgliederzahl benannt. Unter ihnen befinden sich alle parlamentarischen Geschäftsführer der Fraktionen. Neben der Verständigung über den Arbeitsplan des Plenums, beschließt der Ältestenrat über alle inneren Angelegenheiten des Bundestages, soweit sie nicht dem Präsidenten oder dem Präsidium vorbehalten sind. Fachkommissionen, die mit Abgeordneten besetzt sind, bereiten die Beschlüsse des Ältestenrats vor.

In der 15. Wahlperiode (2002-2006) gibt es fünf Kommissionen, eine davon ist die Kommission des Ältestenrates für den Einsatz neuer Informations- und Kommunikationstechniken und -medien, kurz IuK-Kommission[1].

Als Kollegialgremium sind die Mitglieder des Ältestenrats und seiner Kommissionen bemüht, Entscheidungen einvernehmlich zu finden. Trotzdem kann es bei Grundsatzentscheidungen nötig werden, entsprechende Mehrheiten herzustellen.

2.2 IT-Infrastruktur

Die Entscheidungen des Ältestenrats sind für die Abgeordneten und die Verwaltung des Deutschen Bundestages, nicht aber für die Fraktionen, bindend. Diese sind nach dem Fraktionsgesetz vom 11. März 1994 und der Geschäftsordnung des Bundestages rechtlich unabhängig. Sie verfügen über die Entscheidungshoheit für alle ihre Angelegenheiten. Insbesondere haben alle Fraktionen ihre IT-Landschaften unterschiedlich konzipiert. Für ein reibungsloses Funktionieren der elektronischen Kommunikation zwischen Abgeordnetenbüros, Fraktionen und Verwaltung existiert eine Arbeitsgruppe, die mit den IT-Experten aus den verschiedenen Bereichen besetzt ist. Hier werden die strategischen Vorgaben der IuK-Kommission und die IT-Entwicklungskonzepte der Fraktionen koordiniert.

Die Netzinfrastruktur im Bereich des Bundestages folgt den organisatorischen Gegebenheiten. Die äußere Hülle bildet das Intranet des Bundestages.

[1] Vgl. Schick und Schreiner (2003, 22 f.)

Es umfasst:

- das Intranet der Verwaltung des Deutschen Bundestages
- Hier stellen die Bundestagsausschüsse und die Verwaltung Informationen für die Abgeordneten bereit. Auf dieses Intranet haben die Abgeordnetenbüros, die Fraktionen und selbstverständlich die Verwaltung Zugriff.
- die Intranets der einzelnen Fraktion
- Auf dieses Netz haben die Abgeordnetenbüros der entsprechenden Fraktion und die Mitarbeiter der Fraktion Zugriff.
- die „Vertrauensinsel Abgeordnetenbüro"
- Das Abgeordnetenbüro und das zugehörige Wahlkreisbüro bilden eine eigene autonome Organisationseinheit. Die Daten dieser Organisationseinheit werden gegenüber allen Organisationseinheiten mit hohem technischen und organisatorischen Aufwand geschützt.

Im Zuständigkeitsbereich der Bundestagsverwaltung, d.h. auf den Arbeitsplatzrechnern in den Abgeordnetenbüros, der Verwaltung sowie auf den Datei- und Druckservern wird Microsoft Windows NT 4.0 eingesetzt. Im sonstigen Serverbereich der Verwaltung finden sich diverse Betriebssysteme, vereinzelt auch Linux.

Diese IT-Infrastruktur läuft insgesamt stabil. Allerdings hat Microsoft im Jahre 2000 erklärt, den Support für NT 4.0 auslaufen zu lassen. Damit entstand in der Verwaltung der Zwang zu einer Migration der Betriebssystemlandschaft, obwohl diese erst Mitte 1999 mit dem Umzug nach Berlin aufgebaut wurde.

3 Der Migrationsbeschluss des Deutschen Bundestages

Vor dem Hintergrund der Ankündigung Microsofts, den Support für NT 4.0 auslaufen zu lassen, fand sich die Verwaltung des Deutschen Bundestages in der Situation, eine Migration der IT-Infrastruktur durchführen zu müssen. Vor wenigen Jahren noch wäre das gerade aktuelle Betriebssystem von Microsoft auf Arbeitsplatzrechnern und Servern zum Einsatz gekommen. Dies wäre in der trügerischen Annahme geschehen, die bisher getätigten Investitionen im IT-Bereich wenigstens in Teilen zu retten.

Mittlerweile jedoch existiert mit Linux ein lizenzkostenfreies Betriebssystem, das sein Hacker- und Studentenimage schon lange hinter sich gelassen hat. Die SPD-Bundestagsfraktion hat seit 1995 positive praktische Erfahrungen mit Linux gemacht.

Die IuK-Kommission hat deshalb die folgenden Aspekte zur Basis ihrer Empfehlung gemacht:

- Tests mit Linux und Windows in typischen Szenarien
- eine Migrationsstudie
- Überlegungen für die langfristige Entwicklung der IT-Infrastruktur

In den Tests wurde versucht, die typischen Anforderungen an die IT abzubilden, wie sie im täglichen Einsatz im Abgeordnetenbüro existieren. Getestet wurden reine Microsoft-Umgebungen, Szenarien mit Linux-Servern und Windows-Clients und Anordnungen, bei denen sowohl die Server als auch Clients unter Linux betrieben wurden.

Die beiden ersten Modellsysteme waren problemlos zu realisieren. Bei den Linux-Clients zeigte sich, dass die Version 5.2 von StarOffice, deutlich hinter den von Microsoft erhältlichen Office-Programmen zurückblieb. Würde man die Tests mit StarOffice 6.0 bzw. OpenOffice 1.0 wiederholen, wäre diese Einschätzung *wahrscheinlich* nicht mehr haltbar.

Die Interoperabilität zwischen Microsoft Office und StarOffice ist mittlerweile zufriedenstellend. Manchmal setzen beide Systeme die Formatierungen in Dokumenten unterschiedlich um. Das ist in den Fällen, in denen es auch auf die layoutgetreue Wiedergabe ankommt, hinderlich. Ähnliche Probleme haben wir in der Vergangenheit auch verschiedentlich gehabt, wenn wir Dokumente zwischen den unterschiedlichen Word-Versionen austauschen mussten. Wirklich gravierende Probleme zeigen sich jedoch bisher nicht.

Eine Unternehmensberatung wurde mit der Erarbeitung der Migrationsstudie beauftragt. Ausgehend von einer präzisen Analyse des Status Quo wurden die folgenden Migrationsszenarien untersucht:

- Alle Server und Clients werden unter Windows 2000 bzw. XP betrieben. Der Verzeichnisdienst basiert auf ActiveDirectory.
- In der zweiten Variante sollten einige ausgewählte Server, wie zum Beispiel der Mailserver, unter Linux betrieben werden. Die restlichen Server sowie alle Clients laufen unter Windows 2000 oder XP. ActiveDirectory wird wieder als Verzeichnisdienst eingesetzt.
- In der dritten Variante kommt Linux auf den Servern und Windows XP auf den Clients zum Einsatz. Der Verzeichnisdienst wird mit OpenLDAP aufgebaut.
- In der vierten Variante werden sowohl Server als auch Clients vollständig mit Open-Source-Produkten ausgestattet. Auch hier wird OpenLDAP als Verzeichnisdienst verwendet.

Die eigenen Tests haben hauptsächlich die Machbarkeit untersucht. Die Studie hat darüber hinaus den Nutzwert und die Kosten der einzelnen Varianten bewertet. Das Ergebnis gab den beiden gemischten Szenarien den Vorzug. Die Kosten beider Varianten waren gleich hoch – zog man die Unsicherheit in Betracht, die den Kalkulationen anhaftete.

Nachdem aus technischer Sicht eine gemischte IT-Infrastruktur machbar war, hat die IuK-Kommission aus strategischen Überlegungen beschlossen, nur die Clients nach Windows XP zu migrieren. Alle Server hingegen sollten zukünftig mit Linux als Betriebssystem erhalten.

Erste Reaktionen in der Open-Source-Szene in Deutschland haben versucht, dieses Ergebnis in die Nähe eines faulen politischen Kompromisses zu stellen. Die IuK-Kommission ist aber überzeugt, die Basis für eine langfristige IT-Entwicklung im Deutschen Bundestag gelegt zu haben. An deren Ende könnte eine Infrastruktur stehen, die komplett auf OSS aufbaut.

Wäre die Kommission hingegen der Empfehlung der Studie gefolgt, so wären im Serverumfeld wesentliche Anteile auf Basis von Windows 2000 realisiert worden. Insbesondere wäre der Verzeichnisdienst auf ActiveDirectory aufgebaut worden. Gerade letzteres hätte eine stärkere Gewichtung von OSS auf längere Zeit erheblich

erschwert. Eine Diskussion über Linux auf den Clients wäre nicht mehr führbar gewesen.

Diese Option hat sich die IuK-Kommission aber erhalten. Die Entscheidung selbst war nicht einfach, sie musste in einer strittigen Abstimmung herbeigeführt werden. Lediglich die Regierungsfraktionen haben dieser Entscheidung zugestimmt. Die Mitglieder der Oppositionsfraktionen haben sie abgelehnt oder sich enthalten.

Im unmittelbaren Vorfeld der Entscheidung hat die Opposition im Haushaltsausschuss versucht, mit einem Votum des Bundesrechnungshofes die Migration der Server nach Linux zu verhindern. Jedoch hat auch dessen Votum die Absicht der IuK-Kommission unterstützt, sich perspektivisch aus der Abhängigkeit von einem Softwarehersteller lösen zu wollen.

Damit erhält der Beschluss des Bundestages eine Signalwirkung – ganz bestimmt für den Bereich der Öffentlichen Verwaltung. Dort wird man sich zukünftig nicht mehr dafür rechtfertigen müssen, dass man OSS einsetzt. Ein Rechtfertigungszwang entsteht für diejenigen, die bei einer anstehenden Migration OSS nicht in die Prüfung einbeziehen.

4 Hintergründe der Entscheidung

Die IuK-Kommission sieht im Einsatz von OSS eine Neuausrichtung der IT-Strategie im Deutschen Bundestag.

Die Umsetzung der IT-Konzepte, die Mitte bis Ende der 90er Jahre durch Unternehmensberatungen erstellt wurden, haben eine Bindung an die Firma Microsoft zur Folge gehabt. Das bedeutete eine direkte Abhängigkeit von Release-Zyklen und Produktpolitik dieser Firma. Nur so konnte man die Gewähr haben, Support für die im Einsatz befindlichen Systeme zu bekommen. OSS bedeutet im Gegensatz dazu Handlungsfreiheit bei zukünftigen IT-Projekten.

Auf dem Softwaremarkt findet in weiten Teilen nur ein sehr eingeschränkter Wettbewerb statt. Mit OSS hat der Bundestag im Serverbereich auf eine Alternative zu den Produkten einer einzigen Firma zurückgegriffen. Dadurch entsteht etwas mehr Wettbewerb. Wir sind bei den Servern nicht mehr von einer Firma abhängig und können unsere Update-Zyklen zukünftig selbst bestimmen.

In etwa drei Jahren steht die Clientausstattung erneut zur Diskussion. Dann ist die Ausgangslage im Vergleich zum Jahre 2000 ungleich günstiger. Ein Grund hierfür ist, dass im Rahmen der Servermigration auch bereits vorhandene Verfahren migriert werden. Diese Anwendungen werden *zukünftig* web-basiert realisiert. Clientseitig wird nur ein Webbrowser benötigt und die Abhängigkeit von proprietärer Software entfällt.

Der Ältestenrat hat mit seiner Entscheidung für OSS auch das aufgegriffen, was der Entschließungsantrag „Deutschlands Wirtschaft in der Informationsgesellschaft“ fordert. Dieser wurde im Februar 2001 eingebracht und im November des selben Jahres mit den Stimmen der Koalition verabschiedet.

Der Antrag widmet von knapp neun Seiten immerhin eine dem Thema OSS. Zum einen regt er an, den Einsatz von OSS bei Neu- oder Ersatzbeschaffung unter Kostenaspekten zu prüfen. Zum anderen sieht er in OSS eine Chance, den Soft-

waremarkt wiederzubeleben. Gerade auf den Gebieten der Client-Betriebssysteme und der Office-Applikationen hat Microsoft mit einem Anteil von 90% des Marktes eine Vormachtsstellung. Effiziente Kommunikation mit anderen bedingt zunehmend den Einsatz der jeweils aktuellen Versionen dieser Programme. Auf Grund der zunehmenden Komplexität der Programme werden immer höhere Anforderungen an die Hardware gestellt. Wettbewerbsrechtlich ist dieser Zustand problematisch[2].

OSS ist eine besondere Chance für die europäische Softwareindustrie. Zum ersten Mal existiert hier ein Feld, auf dem die USA nicht führend ist. Das damalige Bundesministerium für Wirtschaft und Technologie hat bereits 1999 die Fortentwicklung von Open-Source-Sicherheitskomponenten gefördert. Mittlerweile ist für das Programm kmail ein Plugin entstanden, das die Verwendung von GnuPrivacy-Guard für die Verschlüsselung von Mails gestattet.

Investitionen in OSS können helfen, den eigenen Wirtschaftsstandort zu fördern. So wandern beim Kauf von Microsoft-Produkten die anfallenden Lizenzgebühren zum großen Teil außer Landes. Microsoft beschäftigt in Deutschland nach eigenen Angaben 1500 Mitarbeiter, die hauptsächlich in den Sparten Consulting und Vertrieb arbeiten. Entwickelt werden die Produkte aber in den USA. Mit OSS hat man eine neue Option, Arbeitsplätze gezielt in Deutschland zu schaffen.

Diese Sichtweise auf OSS ist noch wichtiger für Entwicklungsländer. Die Lizenzkosten für Software orientieren sich nicht an dem landesüblichen Preisgefüge, sondern sind weltweit nahezu einheitlich. Finanzschwache Länder können daher nur mit Hilfe von OSS ihren Platz im Informationszeitalter einnehmen.

Bereits ein Jahr vor dem Entschließungsantrag des Bundestages hat der Europäische Rat am 19. und 20. Juli 2000 während einer Tagung in Feira den Aktionsplan „eEurope 2002, eine Informationsgesellschaft für alle" verabschiedet.

Dieser Aktionsplan sieht OSS für die Bereiche:

- Sicheres Internet und intelligente Chipkarte sowie
- Regierung am Netz – elektronischer Zugang zu öffentlichen Diensten

als strategisches Mittel an. Die Mitgliedsstaaten werden aufgefordert, OSS zu fördern.

5 Folgen der Entscheidung

Die Situation in der Öffentlichen Verwaltung beim Einsatz von OSS vor der Entscheidung des Bundestages lässt sich am ehesten mit „Machen, aber nicht darüber reden" beschreiben.

Auf der operativen Ebene existierten etliche Initiativen. Diese waren aber selten Teil einer strategischen Planung. OSS war bis dato für viele Entscheidungsträger im besten Falle eine Randerscheinung aus dem universitären Bereich. Die Behauptung, OSS gefährde den laufenden Betrieb, war nicht zuletzt durch gutes Marketing und Lobbying der Softwarehersteller weit verbreitet.

Das Vorgehen der IuK-Kommission im Vorfeld der Entscheidung hat jedoch gezeigt, dass diese Argumentation jeder Grundlage entbehrt. Die Entscheidung

[2] Vgl. Bundestag (2001).

selbst war ein deutliches Signal. Der Diskussionsprozess der in der Öffentlichen Verwaltung seit der Veröffentlichung des KBSt-Briefs im Frühjahr 2000[3] in Gang kam, hat eine neue Richtung bekommen: Es wird jetzt nicht mehr überlegt, ob man OSS einsetzen kann, sondern wie schnell und in welchem Umfang.

Das Bundesministerium des Innern hat drei Monate nach dem Beschluss des Ältestenrates einen Rahmenvertrag mit IBM abgeschlossen, zu dessen Bestandteilen die Evaluierung vom Pilotprojekten zur Migration nach OSS gehören.

Linux ist seit März 2002 in der Öffentlichen Verwaltung hoffähig geworden. Im Deutschen Bundestag entsteht eine der Referenzinstallationen im Serverbereich, die in Ministerien, Landes- und Kommunalverwaltungen eine ergebnisoffene Diskussion um IT-Strategien möglich macht oder bereits gemacht hat. Schwäbisch-Hall und München sind Beispiele dafür.

Das bedeutet nicht, dass die dort getroffenen Entscheidungen für Linux nur auf Grund der Bundestagsentscheidung gefallen sind. Die Diskussion um OSS im Ältestenrat hat vielmehr katalytisch gewirkt!

Das Engagement der Öffentlichen Verwaltung für OSS ist kein Selbstzweck. Linux darf nicht blind überall dort etabliert werden, wo heute fast ausschließlich Microsoft-Systeme im Einsatz sind. Das hieße eine Monokultur durch eine andere zu ersetzen. Das würde zu ähnlichen Problemen in IT-Sicherheit führen, wie wir sie heute auch schon haben. OSS ist vielmehr ein Weg, sich gegen einzelne Hersteller zu schützen, die ihre Produkt- und Lizenzpolitik mit dem Wissen um die Marktdurchdringung ihrer Produkte betreiben.

6 Open-Source-Software und Offene Standards

Die Ursache des Erfolgs von OSS liegt in ihrer Transparenz. Alle Funktionsprinzipien sind öffentlich dokumentiert. Jeder, der es möchte und der über das nötige Wissen verfügt, kann den Quellcode eines Programms einsehen und verändern.

Aber nicht nur die Programme selbst, auch die Kommunikation zwischen Rechnern beruht auf offenen Standards. Konsequenterweise darf die Verwendung offener Standards nicht nur auf den Bereich von System- und Anwendungsprogrammen beschränkt bleiben. Alle Dateiformate, auch die für Dokumente, müssen auf offenen Standards beruhen.

Ähnlich wie bei proprietären Betriebssystemen, bewirken herstellerspezifische Dokumentenformate Abhängigkeiten. Nachdem die Bundesverwaltung sich beispielsweise vor einigen Jahren auf PDF als Datenaustauschformat geeinigt hatte, hatte der Hersteller eines Programms zur Erzeugung von PDF-Dateien den Preis für die neue Version seiner Software um ein Mehrfaches angehoben. Proteste gegen diese Firmenpolitik können im Einzelfall kleinere Korrekturen bewirken. Auf Grund der Abhängigkeit von einzelnen Herstellern, müssen die neuen Bedingungen in der Regel akzeptiert werden. Ein weiterer Fall war die Änderung der Lizenzpolitik von Microsoft. Gerade hier ist nicht nur der öffentlichen Verwaltung die bereits existierende Abhängigkeit vermittelt worden.

[3] Vgl. KBSt (2000).

Die Ursache für die Bindung an Microsoft wird nicht allein durch das Betriebssystem bedingt, sondern ebenso durch die Office-Programme mit ihrem proprietären Dokumentenformaten. Problematisch ist auch, dass sich Dokumentenformate von Version zu Version ändern können, was fast zwangsläufig zu Neubeschaffungen der Programme führt.

E-Legislation

Ende der letzten Legislaturperiode hat der Ältestenrat beschlossen, dass zukünftig Bundestagsdrucksachen bereits in vorläufiger Fassung in Intranet und in Internet eingestellt werden sollen.

Hintergrund für diese Entscheidung war der Wunsch, der Öffentlichkeit Informationen schneller zur Verfügung zu stellen. Gleichzeitig wurden damit Verfahrensabläufe gestrafft und somit Kosten reduziert. Hier konnte auf Erfahrungen des Schweizer Parlaments zurückgegriffen werden, das bereits vor einigen Jahren vom Papierdokument zum elektronischen Dokument gewechselt ist. Dies hat zu einer deutlichen Reduzierung der Druckkosten geführt.

Die Einstellung von vorläufigen Fassungen in das Internet führte dazu, dass mit der Arbeit an einem Konzept für einen elektronischen Workflow für das Gesetzgebungsverfahren begonnen wurde. Dieses Projekt „E-Legislation" soll bis zum Wechsel zur 16. Legislaturperiode so weit fortgeschritten sein, dass das elektronische Dokument maßgeblich ist. Die Papierform kann dann bei Bedarf abgeleitet werden. Dieses Ziel ist sehr ehrgeizig. Es erfordert eine enge Abstimmung mit allen am Gesetzgebungsverfahren beteiligten, also Bundestag, Bundesregierung, Bundesrat, Bundespräsidialamt und Bundesverfassungsgericht.

Es liegt auf der Hand, dass mit der Umsetzung diesen Konzeptes Arbeitsabläufe sowohl innerhalb des Parlaments aber auch in der Zusammenarbeit von Parlament und Regierung verändert werden müssen. Diverse Probleme müssen gelöst werden:

- Sicherung der Dokumente gegen unberechtigte Manipulation;
- Führen einer lückenlosen Änderungshistorie;
- Einführung der qualifizierten digitalen Signatur, als Ersatz für die persönliche Unterschrift;
- Beachtung der Vorgaben des Grundgesetzes;
- Änderung von Geschäftsordnungen, die ausdrücklich die Papierform erwähnen.

Schließlich wird man auch unter Abgeordneten auf Vorbehalte treffen, die in E-Legislation einen Umbruch in der Parlamentskultur sehen.

Diese Vorhaben steht in einem direkten Zusammenhang zu OSS und Offenen Standards. Es ist heute immer noch möglich eine Gutenbergbibel aus dem Jahre 1456 zu lesen und, soweit vorhanden, mit den Schulkenntnissen in Latein auch zu verstehen. Unwahrscheinlich dagegen ist es im Jahre 2010 Dokumente zu lesen, die 1990 in einem proprietären Dokumentenformat abgespeichert wurden.

Proprietäre Formate sind in der Regel niemals öffentlich dokumentiert worden. Sinn solcher Formate ist es ja geradezu, über mangelnde Transparenz eine Kundenbindung zu erzeugen! Deshalb sind immer ganz spezielle Programme notwendig, um solche Dokumente zu lesen und die darin enthaltenen Informationen wieder

verfügbar zu machen. Releasewechsel der Programme führen nicht selten zu Veränderungen in den Dokumentenformaten. Verschwindet der Hersteller eines Textverarbeitungsprogramms vom Markt, gibt es keine Anpassungen des Programms für neue Betriebssysteme.

Soll vermieden werden, riesige Bestände an Dokumenten regelmäßig in die neueste Version eines proprietären Formats zu wandeln, müssen zusätzlich zu den Programmen die passenden Betriebssysteme vorgehalten werden. Und konsequenterweise müssen auch die alten Computer vorgehalten werden, für die diese Betriebssysteme vor zehn oder zwanzig Jahren entwickelt worden sind. (Das Betriebssystem VMS von der Digital Equipment Corporation ist eben nicht auf einer Pentium-4 Architektur lauffähig.) Diese Aufgabe ist nicht lösbar!

Anders sieht es mit offenen Dokumentenformaten aus. Die Information, die in öffentlich dokumentierten Formaten gespeichert wurde, kann jederzeit wiedergewonnen werden. Auch dann, wenn das Programm nicht mehr verfügbar ist, das ursprünglich zur Erstellung des Dokuments verwendet wurde.

Ein allgemein akzeptiertes, offenes Dokumentenformat wird Auswirkungen auf den Markt der Office-Programme haben. Das Dokumentenformat kann dann nicht mehr dazu genutzt werden, um die Nutzer an einem Wechsel zu einem anderen proprietären oder OSS Produkten zu hindern. Die verschiedenen Anbieter haben vergleichbare Startbedingungen bei der Entwicklung von Software. Es würde zu einer Wiederbelebung des Wettbewerbs kommen. Die Folge wären bessere Produkte, niedrigere Preise und faire Bedingungen für die Nutzer. Vor diesem Hintergrund ist OSS nur das erste Teil in einem Puzzle.

E-Government

Ein weiteres Projekt ist ohne OSS und Offene Standards ebenfalls nicht realisierbar. Aktuell wird viel über E-Government gesprochen. Die Bundesverwaltung hat knapp 400 Dienstleistungen identifiziert, die im Rahmen des Projektes „DeutschlandOnline 2005“ im Internet zur Verfügung gestellt werden sollen.

Die IT-Ausstattung, die in den Haushalten existiert, ist aber unbekannt. Zwei Optionen stehen jetzt zur Wahl:

1. Alle Programme, die zur Nutzung der E-Government-Angebote notwendig sind, werden für die Software des Marktführers optimiert. Dies hat drei Konsequenzen:
   - Ein Teil der Bürger wird von der Nutzung dieser Anwendungen ausgeschlossen oder gezwungen sich diese Software zu kaufen.
   - Bereits bestehende monopolartige Strukturen würden sich weiter verfestigen.
   - Releasewechsel auf der Clientseite machen unter Umständen Änderungen an den Anwendungen in den Behörden notwendig.
2. Die Anwendungen werden so konzipiert, dass keine Einschränkungen für die Clients resultieren. Annahmen über bestimmte Versionen von Betriebssystemen, Office-Programmen oder Browsern werden bei der Entwicklung nicht gemacht.

Variante 1 scheidet schon im Vorfeld aus. Aus Gründen der Akzeptanz von DeutschlandOnline dürfen den Bürgern keine Beschaffungs- oder Lizenzkosten entstehen, wenn sie die Online-Angebote der Verwaltung nutzen.

Die zweite Variante hingegen lässt sich ideal mit OSS realisieren. Die Verwaltung kann diesen E-Government-Client als bootfähige CD-ROM an alle Haushalte in Deutschland verteilen. Niemand wird gezwungen, ein bestimmtes Produkt zu kaufen. Die Schnittstellen und Protokolle auf denen der OSS-E-Government-Client beruht, sind öffentlich dokumentiert und herstellerunabhängig. Damit steht es auch den Entwicklern proprietärer Software frei, ebenfalls Lösungen anzubieten.

7 OSS das Zukunftsmodell?

Der Beschluss des Ältestenrats, Linux als Betriebssystem auf knapp 200 Servern einzusetzen, ist im März 2002 getroffen worden. Ende 2003 werden die ersten Linux-Systeme in den Probebetrieb gehen. Im Herbst 2004 wird die Migration abgeschlossen sein. Es ist abzusehen, dass bald danach die Diskussion um die Ablösung von Windows XP auf den Clients beginnen wird. Es ist reine Spekulation, vorhersagen zu wollen, welches Betriebssystem und welche Anwendungssoftware auf den zukünftigen Clients eingesetzt wird. Ob OSS und eine freie Office-Software auf dem Client machbar sind, müssen die Erfahrungen zeigen, die gegenwärtig in Schwäbisch-Hall oder in München gesammelt werden.

Zwei Vorfälle zeigen aber, dass es bei OSS rechtliche Unklarheiten gibt. SCO hat in den USA u.a. IBM auf Schadensersatz verklagt. IBM wird vorgeworfen, bei Teilen des Linux-Kernels Urheberrechte von SCO verletzt zu haben. In Deutschland hat der Verband der Software-Industrie (VSI) mit einem Gutachten zu OSS überrascht. Es kommt zu dem Schluss, dass der Einsatz von OSS auf Grund der Lizenztypen wie der GPL rechtlich unsicher sei. Es ist möglich, dass sowohl SCO als auch der VSI dem Druck des Marktes erlegen sind. Trotzdem müssen diese Fragen geklärt werden, um das rechtliche Fundament von OSS zu festigen.

Die Ankündigung von Microsoft, den Support für Windows NT 4.0 einzustellen, hat eine Diskussion ausgelöst, die weit über den eigentlichen technischen Aspekt eines Betriebssystemwechsels hinausgeht. Sie hat gezeigt, dass in dem strategisch wichtigen Marktsegment für Betriebssysteme und Office-Programme der Wettbewerb schleichend außer Kraft gesetzt wurde.

Mit OSS existiert aber mittlerweile eine Basis, auf der sich eine herstellerunabhängige IT-Strategie entwickeln lässt. Das heißt nicht, dass sich proprietäre Software überlebt hat oder *a priori* OSS unterlegen wäre. Das Ziel ist vielmehr, eine Infrastruktur zu schaffen, die ein Nebeneinander von proprietärer und quelloffener Software gestattet. Die Offenlegung von Schnittstellen und Dokumentenformaten ist ein erster Schritt auf diesem Weg. Der nächste wäre die Einigung von Herstellern und Entwicklern auf offene Standards. Die konsequente Einhaltung offener Standards sollte daher zu einem Qualitätsmerkmal für Software werden.

Literatur

Bundestag (2001): *Deutschlands Weg in die Informationsgesellschaft*, Bundestagsdrucksache 14/5246

KBSt (2000): *Open Source Software für die Bundesverwaltung*, online http://www.kbst.bund.de/Themen-und-Projekte/Software-74/OSS.htm

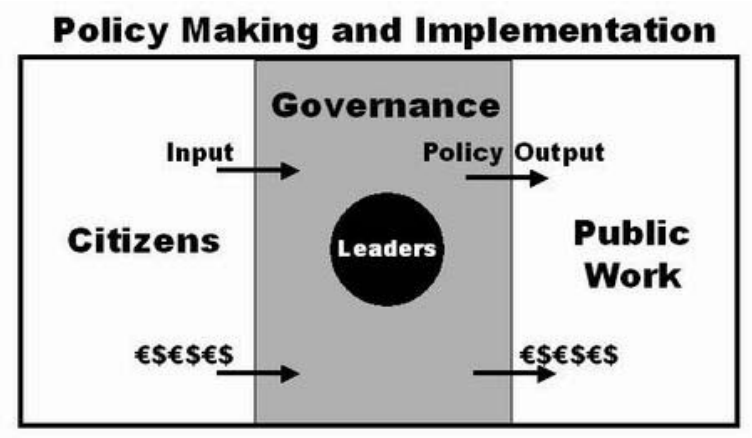
Schick, R., H.J. Schreiner (2003): *So arbeitet der Deutsche Bundestag*, Rheinbreitbach

E-Democracy, E-Governance and Public Net-Work

Steven Clift

Introduction*

While the art and practice of government policy-making, citizen participation, and public work is quite complex, the following illustration provides a simple framework used in this paper:



In this model of traditional government policy-making:

- Citizens provide occasional input between elections and pay taxes.
- Power in the Governance infrastructure is centered with political leaders who determine broad policy priorities and distribute resources based on those priorities and existing programs and legal requirements.
- Through government directly, and other publicly funded organizations, Public Work represents the implementation of the policy agenda and law.

Over time of course, bureaucratic barriers to reform make it difficult for leaders to recognize changes in citizen needs and priorities. Citizen input, outside of elections, often has a difficult time getting through. Disconnects among citizens, leaders, and those who implement public work are often based on the inability to easily communicate through and across these groups.

As our one-way broadcast world becomes increasingly two-way, will the governance process gain the ability to listen and respond more effectively?

---

* Wir danken dem Autor für die Genehmigung zum Abdruck des Aufsatzes. Der Beitrag wurde ursprünglich im Internet unter http://www.publicus.net veröffentlicht.

The information-age, led by Internet content, software, technology, and connectivity, is changing society and the way we can best meet public challenges. E-democracy, e-governance, and public net-work are three interrelated concepts that will help us map out our opportunity to more effectively participate, govern, and do public work.

E-Democracy

E-democracy is a term that elicits a wide range of reactions. Is it part of an inevitable technology driven revolution? Will it bring about direct voting on every issue under the sun via the Internet? Is this just a lot of hype? And so on. (The answers … no, no, and no.)

Just as there are many different definitions of democracy and many more operating practices, e-democracy as a concept is easily lost in the clouds. Developing a practical definition of E-Democracy is essential to help us sustain and adapt everyday representative democratic governance in the information age.

Definition

After a decade of involvement in this field, I have established the following working definition:

E-Democracy is the use of information and communications technologies and strategies by "democratic sectors" within the political processes of local communities, states/regions, nations and on the global stage.

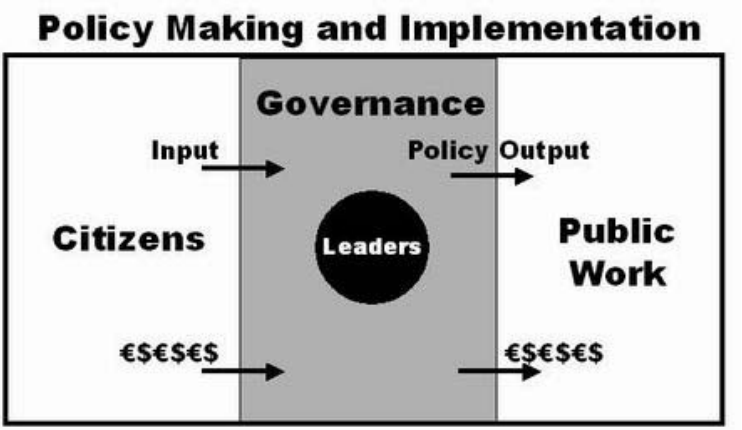
The "democratic sectors" include the following democratic actors:

- Governments
- Elected officials
- Media (and major online Portals)
- Political parties and interest groups
- Civil society organizations
- International governmental organizations
- Citizens/voters

Current E-Democracy Activities

Each sector often views its new online developments in isolation. They are relatively unaware of the online activities of the other sectors. Those working to use information and communication technologies (ICTs) to improve or enhance democratic practices are finding e-democracy a lot more challenging to implement than speculating on its potential. This is why it is essential for the best e-democracy lessons and practices to be documented and shared.

This simplified model illustrates e-democracy activities as a whole. Building on the first diagram, it sits as a filter on the "input" border between citizens and governance in first diagram:



Governments provide extensive access to information and interact electronically with citizens, political groups run online advocacy campaigns and political parties campaign online, and the media and portal/search sites play a crucial role in providing news and online navigation. In this model, the "Private Sector" represents commercially driven connectivity, software, and technology. This is the whole of e-democracy.

E-democracy is not evolving in a vacuum with these sectors only. Technology enhancements and online trends from all corners of the Internet are continuously being adopted and adapted for political and governance purposes. This is one of the more exciting opportunities as e-mail, wireless networking, personalization, weblogs, and other tools move in from other online content, commerce, and technology areas and bring innovation and the opportunity for change with them.

Looking to the center of model, the only ones who experience "e-democracy" as a whole are "citizens." In more "wired" countries most citizens are experiencing information-age democracy as "e-citizens" at some level of governance and public life. In developing countries, e-democracy is just as important, but exists as more of an institution-to-institution relationship. In all countries, the influence of "e-democracy" actually reaches most of the public through its influence on the traditional media and through word of mouth via influential members of the community.

"E-Citizens" – Greater Citizen Participation?

To many, e-democracy suggests greater and more active citizen participation enabled by the Internet, mobile communications, and other technologies in today's representative democracy. It also suggests a different role for government and more participatory forms of direct citizen involvement in efforts to address public challenges. (Think e-volunteerism over e-voting.)

Some take this further and view the information revolution as an inherently democratic "disruptive technology" that will dramatically change politics for the better. This view has diminished considerably, as existing democratic actors have demonstrated their ability to incorporate new technologies and online communication

strategies into their own activities and protect their existing interests. They have to in order to survive.

In the future, most "e-democracy" development will naturally result from ICT-accelerated competition among the various political forces in society. We are experiencing a dramatic "e-democracy evolution." In this evolution, the role, interests, and the current and future activities of all actors is not yet well understood. There is still an opportunity to influence its development for the better.

Things will change, but as each democratic sector advances their online activities, democratic intent will be required to achieve the greater goals of democracy[1].

E-Governance

I use the phrase "Representative E-Government" to describe the e-democracy activities of government institutions. Others call this "e-governance." Whether a local government or a United Nations agency, government institutions are making significant investments in the use of ICTs in their work. They are expressing "democratic intent." Their efforts make this one of the most dynamic and important areas of e-democracy development.

There are distinct differences in how representative institutions and elected officials use ICTs compared to administrative agencies and departments. The use of ICTs by parliaments, heads of state/government, and local councils (and elected officials in these institutions) lags significantly behind the administrative-based e-government service and portal efforts. This is a services first, democracy later approach.

This focus of e-government resources on services does not mean that e-democracy is not gaining increased attention in some governments. In fact, leading e-service governments are now at a point where they are exploring their e-democracy responsibilities more seriously.

Goals for E-Democracy in Governance

Investment in traditional e-government service delivery is justified based on the provision of greater citizen convenience and the often-elusive goal of cost-savings. Goals for e-government in governance that promote democracy and effective governance include:

- Improved government decisions
- Increased citizen trust in government
- Increased government accountability and transparency
- Ability to accommodate the public will in the information-age
- To effectively involve stakeholders, including NGOs, business, and interested citizen in new ways of meeting public challenges (see public net-work below)

[1] Related resources: http://www.publicus.net/articles/edemresources.html; http://www.publicus.net/articles/future.html; http://www.publicus.net/ebook.

Consultation Online

The first area of government e-democracy exploration has focused on consultation within executive policy-making processes. Governments, like the United Kingdom and Canada, are taking their consultative frameworks and adapting them to the online environment. New Zealand and Canada now have special portals dedicated to promote the open consultations across their governments. This includes traditional off-line opportunities as well as those where online input is encouraged. Across the UK, a number of "online consultations" have been deployed to gather special citizen input via the Internet.

Examples:

- Consulting Canadians[2]
- New Zealand – Participate[3]
- UK E-Democracy Consultation[4]
- Others, including hosting and best practice tips[5]

Accountability, Trust, the Public Will

These three themes are emerging on the e-democracy agenda. Building government accountability and transparency are a significant focus of e-government in many developing countries. E-government is viewed an anti-corruption tool in places like South Korea, Mexico, and others. Trust, while an important goal, can only be measured in the abstract. Establishing a causal relationship between e-government/e-democracy experiences and increased levels of trust will be difficult.

Ultimately, the main challenge for governance in the information age will be accommodating the will of the people in many small and large ways online. The great unknown is whether citizen and political institutional use of this new medium will lead to more responsive government or whether the noise generated by competing interests online will make governance more difficult. It is possible that current use of ICTs in government and politics, which are often not formulated with democratic intent, will actually make governance less responsive.

One thing is clear, the Internet can be used to effectively organize protests and to support specific advocacy causes. Whether it was the use of e-mail groups and text messaging protesting former President Estrada of the Philippines or the fact a majority of Americans online sent or received e-mail (mostly humor) after the Presidential election "tie" in the United States, major moments in history lead to an explosion of online activity. The social networks online are very dynamic and governments need to be prepared to accommodate and react to "electric floods." When something happens that causes a flood, people will expect government to engage them via this medium or citizens will instead view government as increasingly unresponsive and disconnected with society they are to serve[6].

---

2 Http://www.consultingcanadians.gc.ca.

3 Http://www.govt.nz/en/participate.

4 Http://www.e-democracy.gov.uk.

5 Http://www.publicus.net/articles/consult.html.

6 For more on the e-government and democracy, watch for the 2003 United Nations World Public Sector Report. Details will be shared on DoWire: http://www.e-democracy.org/do.

Top Ten E-Democracy "To Do List" for Governments Around the World

1. Announce all public meetings online in a systematic and reliable way. Include the time, place, agenda, and information on citizen testimony, participation, or observation options. Use the Internet to build trust in in-person democracy.
2. Put a "Democracy Button" on your site's top page which brings them to a special section detailing the agencies/government units purpose and mission, top decision-makers, links to enabling laws, budget details and other accountability information. Share real information that help a citizen better understand the legitimacy of your government agency and powers. Give citizens real information on how to best influence the policy course of the agency. This could include links to the appropriate parliamentary or local council committees and bodies.
3. Implement "Service Democracy." Yes, most citizens simply want better, more efficient access to service transactions and information products your agency produces. Learn from these relationships. Actively use comment forms, online surveys, citizen focus groups to garner the input required to be a responsive e-government. Don't automate services that people no longer want or need. Use the Internet to learn about what you can do better and not just as a one-way self-service tool designed to limit public interaction and input.
4. End the "Representative Democracy Online Deficit." With the vast majority of government information technology spending focused on the administrative side government, the representative institutions from the local level on up to the Federal government are growing increasingly weak. Invest in the technology and communications infrastructure of those institutions designed to represent the people. Investing in elected officials' voice through technology is investing in the voice of the people. Cynicism aside, options for more direct democracy can be explored, but invest in what we have today – representative democracy.
5. Internet-enable existing representative and advisory processes. Create "Virtual Committee Rooms" and public hearings that allow in-person events to be available in totality via the Internet. Require in-person handouts and testimony to be submitted in HTML for immediate online availability to those watching or listening on the Internet or via broadcasting. Get ready to datacast such items via digital television. Encourage citizens to also testify via the Internet over video conferencing and allow online submission of written testimony. The most sustainable "e-democracy" activities will be those incorporated into existing and legitimate governance processes.
6. Embrace the two-way nature of the Internet. Create the tools required to respond to e-mail in an effective and timely manner. E-mail is the most personal and cherished Internet tool used by the average citizen. How a government deals with incoming e-mail and enables access to automatic informational notices based on citizen preferences will differentiate popular govern-

ments from those that are viewed as out of touch. Have a clear e-mail response policy and start by auto-responding with the time and date received, the estimated time for a response, what to do if none is received, and a copy of their original message. Give people the tools to help hold you accountable.

7. Hold government sponsored online consultations. Complement in-person consultations with time-based, asynchronus online events (one to three weeks) that allow people to become educated on public policy issues and interact with agency staff, decision-makers, and each other. Online consultations must be highly structured events designed to have a real impact on the policy process. Don't do this for show. The biggest plus with these kinds of events is that people may participate on their own time from homes, schools, libraries and workplaces and greater diversity of opinions, perspectives, and geography can increase the richness of the policy process. Make clear the government staff response permissions to allow quick responses to informational queries. Have a set process to deal with more controversial topics in a very timely (24–48 hours) fashion with direct responses from decision-makers and top agency staff. Do this right and your agency will want to do this at least quarterly every year,, do it wrong the first time and it will take quarter of a century to build the internal support for another try. Check on the work in Canada, The Netherlands, Sweden and United Kingdom in particular and you'll discover government that are up to some exciting work.
8. Develop e-democracy legislation. Tweak laws and seek the budgetary investments required to support governance in information age. Not everything can be left voluntary – some government entities need a push. What is so important that government must be required to comply? There is a limit to what can be squeezed out of existing budgets. Even with the infrastructure in place the investment in the online writers, communicators, designers, programmers, and facilitators must be increased to make Internet-enhanced democracy something of real value to most citizens and governments alike.
9. Educate elected officials on the use of the Internet in their representative work. Get them set-up technologically and encourage national and international peer-to-peer policy exchanges among representatives and staff. Be careful to prevent use this technology infrastructure for incumbency protection. Have well designed laws or rules to prevent use of technology and information assets in unknown ways. Don't be overly restrictive, but e-mail gathered by an elected official's office shouldn't suddenly be added to a campaign e-mail list. Be sure the tell them to read the "Top Ten Tips for Wired Elected Officials".
10. Create open source democracy online applications. Don't waste tax dollars on unique tools required for common governmental IT and democracy needs. Share your best in-house technology with other governments around the world. Leverage your service infrastructure, be it proprietary or open source, for democratic purposes. With vast resources being spent on making administrative government more efficient, a bit of these resources should be

used "inefficiently." Democracy is the inefficiency in decision-making and the exercise of power required for the best public choices and outcomes. Even intentional democratic inefficiency can be made more effective with IT.

Top Ten Tips for "Weos" – Wired Elected Officials

1. Use the Internet to communicate.
   Whether it is private one-to-one or public group communication, interaction is the most transformative and powerful political application on the Internet. Speech on the Internet is meaningless unless there is free electronic association.
2. Use the Internet to disseminate information.
   Whether as part of your official duties or party/campaign work, encourage your constituents or political supporters to join your one-way e-mail list(s). The web is passive from an organizers perspective because people rarely visit the same site twice. You want people to join or "opt-in" to your e-mail lists so you can share your message widely little or no cost.
3. Develop multiple e-mail address identities on the Internet.
   Have one e-mail address for public official constituent communication, one internal address for official government work, and at least one personal e-mail address for unofficial campaign/party political communication and other personal communication.
4. Promote "E-Democracy" within your existing representative structures to enable "wired" public participation.
   Take your existing processes such as committee hearings, public testimony, constituent communication and adapt them to the information age. Active integration of information and communication technology into legally representative democracy is essential to maintain legitimacy and improve democracy. Pass model "E-Democracy laws" that require representative and consultative features of the administrative side of government and other government bodies to be fully accessible online. Start by requiring that all public meeting notices and agendas be posted online through a uniform system.
5. Use the Internet to connect with peers around the world.
   The Internet is a terrific way to establish intentional and value-added opportunities for peer-to-peer information sharing among people with similar interests or goals. Take any public policy topic of interest and create networks for you and your staff. Don't wait for others to build global policy network of elected officials. Become a known global expert in a topic area by taking the initiative now.
6. Use the Internet to access information.
   It is an information maze out there. Be patient and you will often find what you need. Use your peer connections and assist each other with research requests and needs. Sending a query to the group will often result in references to useful information just as proactively sharing the results of your online research will provide value to others. Think of this as "just-in-time-democra-

cy" through the use of your expert and other's online "best practitioner" networks.

7. Use the Internet to access information smartly.
   Settle on a search engine like Google[7] and subject trees like the Open Directory[8] and Yahoo[9]. Learn how they work. Find similar sites by reverse searching – for example "link: http://www.e-democracy.org" will find all pages indexed at Google linking to that page. Try the reverse search to find our who links to your site.
8. Use the Internet to be fed information automatically.
   Subscribe to select e-mail newsletters and announcements list on the web sites you find most useful. Let them tell you when they have something new. Use e-mail filtering (ask your technical staff for help) to sort your incoming e-mail into different folders to keep e-mail list messages separate from e-mail sent personally to you.
9. Use the Internet for intelligence.
   Whether it is a site you find useful or the site of your political opponents, use the Internet to monitor their public activities and documents. You can use tools like Spy On It[10] to set automatic page watchers that will notify you when something new is posted on a web site. Some of the best public policy information is not promoted beyond placement on a web page. Let a web reminder tell you something has been changed or added.
10. Promote integrated services for all elected officials across the organization.
    Uniform systems, networks, and equipment should be overhead covered by the representative institution itself and not a cost to members directly (at least for the essential technology base). This is a balance of power issue. If the administrative side of government invests billions in their information infrastructure, the representative side must invest as well to remain a relevant voice for an increasingly wired society. The same goes for those in political party based elections – promote an integrated and aggregated campaign information infrastructure that may be used securely and strategically by all party candidates.

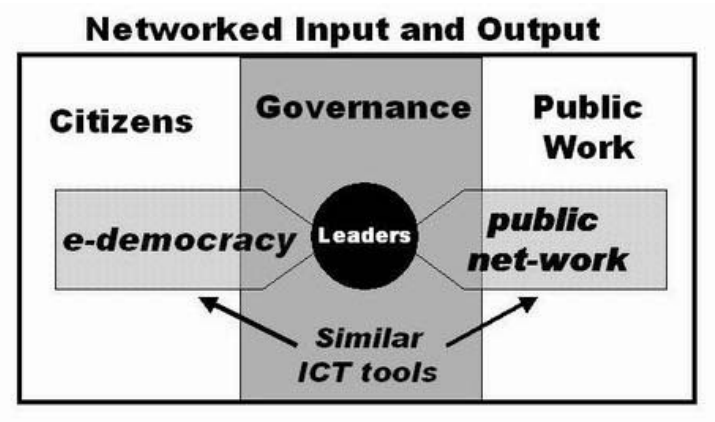
Public Net-Work

Public net-work is a new concept. It represents the strategic use of ICTs to better implement established public policy goals and programs through direct and diverse stakeholder involvement online.

If e-democracy in government represents input into governance, then public net-work represents participative output using the same or similar online tools. Public net-work is a selective, yet public, approach that uses two-way online information exchange to carry out previously determined government policy.

7 Http://google.com.
8 Http://dmoz.org.
9 Http://yahoo.com.
10 Http://spyonit.com.

Building on the first diagram, the following "bow-tie" model suggests a more fluid communication environment that can be used to bring citizens and public work stakeholders closer to the center of governance. It also suggests that policy leaders can reach out and develop closer relationships with citizens and stakeholders.



What are public net-work projects?

Public net-work projects have the following things in common:

1. They are designed to facilitate the online exchange of information, knowledge and/or experience among those doing similar public work.
2. They are hosted or funded by government agencies, intergovernmental associations, international government bodies, partnerships involving many public entities, non-governmental organizations, and sometimes foundations or companies.
3. While they are generally open to the public, they are focused on specific issues that attract niche stakeholder involvement from other government agencies, local governments, non-governmental organizations, and interested citizens. Essentially any individual or group willing to work with the government to meet public challenges may be included. However, invite-only initiatives with a broader base of participants are very similar to more strictly defined "open" public net-work initiatives.
4. In a time of scare resources, public net-work is designed to help governments more effectively pursue their established missions in a collaborative and sustainable manner.

In order to work, public net-work initiative hosts need to shift from the role of "top experts" or "sole providers" of public services to facilitators of those working to solve similar public problems. Public net-work moves beyond "one-way" information and service delivery toward "two-way" and "many-to-many" exchange of information, knowledge, and experience.

Features

Publicly accessible public net-work projects currently use a mix of ICT tools available. The successful projects adopt new technologies and strategies on an incremental trial and error basis. Unleashing all of the latest tools and techniques without a user base may actually reduce project momentum and user participation.

To succeed, these projects must adapt emerging models of distributed information input and information sharing and develop models for sustained knowledge exchange/discussion. They must also build from the existing knowledge about online communities, virtual libraries, e-newsletters, and Communities of Practice/Interest.

Some of the specific online features include:

1. Topical Portal – The starting point for public net-work is a web site that provides users a directory to relevant information resources in their field – these often include annotated subject guide links and/or standard Yahoo-style categories.
2. E-mail Newsletter –. Most projects keep people up-to-date via regularly produced e-mail newsletters. This human edited form of communication is essential to draw people back to the site and can be used to foster a form of high value interaction that helps people feel like they are part of the effort.
3. Personalization with E-mail Notification – Some sites allow users to create personal settings that track and notify them about new online resources of interest. New resources and links to external information are often placed deep within an overall site and "What's New" notification dramatically increases the value provided by the project to its users.
4. Event Calendar – Many sites are a reliable place to discover listings of key current events and conferences.
5. FAQ and Question Exchange – A list of answers to frequently asked questions as well as the regular solicitation of new or timely questions from participants. Answers are then gathered from other participants and shared with all via the web site and/or e-newsletter.
6. Document Library – Some sites move beyond the portal directory function and gather the full text of documents. This provides a reliable long-term source of quality content that often appears and is removed from other web sites without notice.
7. Discussions – Using a mix of e-mail lists and/or web forums, these sites encourage ongoing and informal information exchange. This is where the "life" of the public net-work online community is often expressed.
8. Other features include news headline links from outside sources, a member directory, and real-time online features.

Examples

- CommunityBuilders New South Wales[11]
- International AIDS Economics Network[12]

[11] Http://www.communitybuilders.nsw.gov.au.

[12] Http://www.iaen.org.

- OneFish[13]
- DevelopmentGateway[14]
- Research Institute of Economy, Trade and Industry – Digital New Deal[15]
- UK Improvement and Development Agency – Knowledge[16]

Lessons

1. Government partnerships, with their public missions and resources, often make ideal hosts for broad, horizontal information exchange. Government departments that feel their status/purpose will be threatened by shifting from an expert gatekeeper to an involved facilitator are not ideal hosts.
2. All online features must be designed with the end user in mind. They must be usable and easy to learn. Complex systems reduce the size of the participatory audience – public net-work cannot rely on an internal office environment where people are required to learn new systems or use specialty software beyond e-mail and a web browser. To provide a strong incentive, these systems must save the time it takes those implementing public policy to do their job effectively.
3. Public net-work sites broaden the awareness of quality information resources on a timely basis. Finding what you need, when you need it is more likely to occur when a community of interest participates in building a comprehensive resource. However, over time these sites will naturally face currency issues that must be handled. There are limits to the value of information exchange. Too much information, or bad information, can paralyze decision-making or distract people from the task at hand. All good things should be taken in moderation.
4. Building trust among the organizations and individuals participating in the development and everyday use of a collaborative site is essential. This relates to developing the "neutral host" facilitation role, along with sustained funding, by the host. Special care must be taken when building partner relationships and host "branding" kept to a minimum. Partnerships, with clear responsibilities and goals, will better position efforts as a truly participatory community projects.
5. Gathering and sharing incentives, particularly for resource links is a particularly tricky area. Involving people with solid librarianship and communication skill sets is essential. Creating a more sustainable model where participants more actively submit information (e.g. seeking submissions from users for more than 5% of link listings for example) is an ongoing challenge. In-kind partnerships where staff time is donated may be more effective than relying on the time of unaffiliated individual volunteers. With more localized efforts, individual volunteers may be the best or only option.

13 Http://www.onefish.org.
14 Http://www.developmentgateway.org.
15 Http://dnd.rieti.go.jp.
16 Http://www.idea-knowledge.gov.uk.

6. Informal information sharing has tremendous potential. To effectively encourage horizontal communication, facilitation is often required. Projects must leverage existing online communities and be willing to use technologies, like e-mail lists if that is what people will actually use. In my opinion, the CommunityBuilder.NSW site is one of the few sites that effectively integrate e-mail and web technology to support sustained online deliberation and information exchange.
7. The connection to decision-makers and authority is significant. Government-led public net-work projects require political leadership and strong management support. Paradoxically, an effective online involvement program on the implementation side of government, if connected to government leaders, may operate as an "early warning system" and allow government to adapt policy with fewer political challenges[17].

Conclusion

To be involved in defining the future of democracy, governance and public work at the dawn of the information-age is an incredible opportunity and responsibility. With the intelligent and effective application of ICTs, combined with democratic intent, we can make governments more responsive, we can connect citizens to effectively meet public challenges, and ultimately, we can build a more sustainable future for the benefit of the whole of society and world in which we live.

[17] The public net-work section above is based on an article I wrote for the OECD's E-Government Working Group. An expanded discussion of case examples and the future direction of public net-work is available in Public Net-work: Online Information Exchange in the Pursuit of Public Service Goals: http//www.publicus.net/articles/oecdpublicnetwork.rtf.

Kapitel 5

Gesellschaft

Einleitung

ROMAN BÜTTNER

> XVII. La propriété étant un droit inviolable et sacré, nul ne peut en être privé, si ce n'est lorsque la nécessité publique, légalement constatée, l'exige évidemment, et sous la condition d'une juste et préalable indemnité.
>
> *La Déclaration des Droits de l'Homme et du Citoyen, 1789* [1]

I. Die Ökonomie des Tausches

Was hat ein Kapitel Gesellschaft in einer Publikation mit dem so technisch anmutenden Titel „Open-Source-Jahrbuch" zu suchen? Für die Antwort auf diese Frage soll zunächst auf einige Charakteristika der modernen Wirtschaftsweise hingewiesen werden.

Mit der Entwicklung der Industriegesellschaften im ausgehenden 18. Jahrhundert etablierte sich als umfassendes Prinzip allen ökonomischen Handelns etwas, was in agrarisch geprägten Gesellschaften nur eine Nebenrolle gespielt hat: der Markt. Die Marktwirtschaft wurde zur Basis der europäischen Gesellschaften. Als Rahmen, der das Funktionieren dieser neuen Wirtschaftsweise gewährleistet – als Überbau – setzte sich die bürgerliche Demokratie durch, weil sie zwei wesentliche Grundbedingungen der Marktwirtschaft garantiert: die Freiheit eines jeden, zu tun und zu lassen, was er will, solange er damit nicht die Freiheit eines anderen einschränkt, und die (theoretische) Gleichheit eines jeden vor der Obrigkeit, vor dem Recht und in den Chancen.

Schon die Verfasser der Menschen- und Bürgerrechte, die im Rahmen der Französischen Revolution erstmals verkündet wurden, wussten, dass das zentrale Element einer Marktwirtschaft das Eigentum sein muss. Seither wird in allen bürgerli-

[1] Artikel XVII der Erklärung der Menschen- und Bürgerrechte: „Da das Eigentum ein unverletzliches und geheiligtes Recht ist, kann es niemandem genommen werden, es sei denn, dass die gesetzlich festgestellte öffentliche Notwendigkeit dies eindeutig erfordert und vorher eine gerechte Entschädigung festgelegt wird." Download der Originalfassung: http://www.elysee.fr/instit/text1.htm, deutsche Übersetzung: http://www.elysee.fr/all/instit/text1_.htm.

chen Verfassungen Eigentum als (nahezu) unantastbar definiert, denn nur solange das Individuum über sein Eigentum in jeglicher Hinsicht frei verfügen kann, ist ein funktionierender Markt möglich.

Das Prinzip der Marktwirtschaft ist – oberflächlich betrachtet – vergleichsweise einfach: Benötigt der Mensch etwas, trägt er sein Eigentum zum Markt und tauscht etwas davon ein. Wie viele Güter er für den Erwerb eines fremden Gutes aufwenden muss, wird von Angebot und Nachfrage bestimmt. Der Preis wird ausgehandelt und ist somit zunächst weitgehend unabhängig von den Kosten, die die Bereitstellung eines Gutes hat. Die Konkurrenz mehrerer Anbieter (wo sie vorhanden ist) stellt dann einen Preis sicher, der den Kosten ungefähr entspricht.

Um den Handel auf dem freien Markt nicht übermäßig zu komplizieren, bedurfte es noch eines weiteren Elementes, dessen Erfindung allerdings schon jahrtausendealt war: der des Geldes. Geld bildet ein Äquivalent, gewissermaßen ein Modell von Eigentum. Es hat aber noch einen weiteren Vorteil. Eigentum wird in den gängigen Theorien der Aufklärung erworben, indem man sich „freie" Güter (also solche, die keinen Eigentümer haben) aneignet oder indem man Güter durch Arbeit verändert. Letzteres bedarf des Einverständnisses des Eigentümers. Da aber Arbeit unvermeidbar ist (kaum jemand kann etwas mit einem Klumpen Eisenerz anfangen), hat der Eigner (des Erzklumpens), der nicht alle erforderliche Arbeit allein leisten kann, ein Interesse daran, dass ein etwaiger Mitarbeiter keinen Anspruch auf das Produkt seiner Arbeit (bspw. ein Automobil) erheben kann. Er entschädigt seinen Mitarbeiter daher mit dem Eigentumsäquivalent Geld, wobei dieser alle Rechte am Produkt abtreten muss. So wird es möglich, dass auch Individuen, die über keinerlei Eigentum verfügen, an der Marktwirtschaft teilnehmen (können und müssen). Sie bieten auf dem Markt statt ihres Eigentums ihre Arbeitskraft an und erwerben anstelle von Eigentum Geld.

In der Natur existieren zwei wesentliche Vorbedingungen für einen funktionierenden Markt: Güter sind nur in endlicher Anzahl vorhanden, und sie werden verbraucht. Ein Bäckereibetrieb kann mit einem festgelegten Kostenaufwand nur eine beschränkte Anzahl von Broten herstellen, eine Automobilfabrik ebenso nur eine kleine Anzahl Autos. Somit ist das Angebot an materiellen Gütern immer mehr oder weniger „knapp". Da ein Brot aber aufgegessen wird und ein Auto verschleißt, also nur durch Arbeit wieder in einen funktionierenden Stand versetzt werden kann, ist sichergestellt, dass immer neue Nachfrage besteht. Durch ein prinzipiell endliches Angebot und eine Nachfrage, die größer als null ist, wird das marktübliche Preisbildungssystem in einer Industriegesellschaft sinnvoll.

Wie sieht das aber in der so genannten Informationsgesellschaft aus? Der Begriff „Informationsgesellschaft" als Gegensatz zur „Industriegesellschaft" ist ein relativ neuer Terminus, mit dem versucht wird, die digitale Welt in bekannte Schemen einzuordnen. Seit den 80er Jahren des letzten Jahrhunderts wird ein immer größerer Teil des vorhandenen Kapitals nicht mehr in der klassischen Produktion materieller Güter akkumuliert, sondern im Glücksspiel der „neuen Märkte" verloren und gewonnen. Der Handel mit „Information", „Wissen", „Ideen" gewinnt zunehmend an Bedeutung. Die Güter dieser neuen Welt sind nicht mehr materiell (wie Brote und Autos), sie sind Produkte des Geistes – Algorithmen, Argumentationen, Darstellun-

gen. Das Wesentliche aber bleibt: Marktwirtschaft als Basis und bürgerliche Demokratie im Überbau.

Mangels eines neuen Modells wurde die beschriebene Verwertungslogik der Industriegesellschaft auf die neuen Märkte nahezu unverändert übertragen. Nach dem Prinzip von Angebot und Nachfrage wird mit Informationsgütern gehandelt. Entsprechend dem materiellen Eigentum gibt es ein *geistiges Eigentum* (im anglo-amerikanischen Sprachgebrauch: *intellectual property*), dessen Schutz in allen Ländern der westlichen Hemisphäre durch das Patentrecht, den Marken- und Musterschutz sowie den Urheberrechtsschutz (im anglo-amerikanischen mit kleinen inhaltlichen Differenzen: *copyright*) garantiert wird.

Spätestens seit der Verbreitung elektronischer Medien und des Internets, also der Digitalisierung von Informationsgütern, zeigen sich aber die Probleme der alten Verwertungslogik. Solange geistiges Eigentum noch an endliche und sich verschleißende Trägermedien gebunden war (Buchseiten, Schallplatten, Disketten etc.), ließ sich dies verdrängen. Ein Informationsgut (ein Buch, ein Stück Software) enthielt neben den Fixkosten (Salär des Autors, Programmierers) auch Grenzkosten (Kosten für die Herstellung des einzelnen Buches, der Diskette mit dem Computerprogramm). Durch die digitale Vernetzung werden Informationsgüter aber von jeglichen materiellen Trägermedien gelöst, der Anteil der Grenz- an den Gesamtkosten sinkt auf fast null. Im digitalen Netz sind Informationsgüter nahezu unendlich oft kopierbar, ohne dass sich die Gesamtkosten nennenswert erhöhen und ohne dass sich ihre Qualität und ihr Inhalt verändern.

Eine weitere Eigenschaft digitaler Güter verschärft das Problem: Sie verschleißen nicht. Ein Computerprogramm aus der Pionierzeit der 60er Jahre funktioniert heute genau so wie bei seiner Erstellung. Wenn sich also – wie man anhand des Beispiels der Industriellen Revolution erwarten kann – das Tempo der Weiterentwicklung in der digitalen Welt verlangsamt, sieht sich der Benutzer digitaler Güter seltener veranlasst, Neuerwerbungen zu tätigen.

Der Unterschied zwischen materiellen Produkten und Informationsgütern, nämlich dass es sich bei den Ersten um einzelne, jeweils Kosten verursachende Stücke, bei Letzteren aber lediglich um Kopien eines „Ursprungsgutes" handelt, wird in der digital vernetzten Welt besonders deutlich. Insofern bleibt fraglich, ob die gängigen Theorien zu materiellem Eigentum überhaupt auf das geistige übertragen werden können.

Da digitale Güter wegen ihrer verschwindend geringen Grenzkosten theoretisch in nahezu unendlicher Anzahl vorhanden sind und sich nicht verbrauchen, gerät das Prinzip „Angebot und Nachfrage" selbst in einem optimistischen Szenario, in dem wenigstens immer eine Nachfrage besteht, ins Wanken. Digitale Güter sind keine klassischen Konsumgüter mehr, sondern „Partizipationsgüter", an denen der Nutzer teilhat, ohne sie dabei zu verbrauchen (vgl. Zappe 2003). Beim Erwerb eines solchen erhält man nicht das Original, sondern eine identische Kopie, dem Anbieter geht also nichts (außer einem potenziellen Käufer) verloren.

Um eine funktionierende Ökonomie nach dem klassischen Modell auch in der virtuellen Welt zu ermöglichen, muss das Angebot von Kopien eines Informationsgutes künstlich verknappt werden. Das geschieht durch rechtliche Schranken wie

das Patent- und das Urheberrecht – gerade Letzteres ist in den letzten Jahren in Nordamerika und Europa erheblich verschärft worden.

Das Missverhältnis zwischen materiellem und geistigem Eigentum tritt derzeit an verschiedenen Stellen zu Tage. Einerseits ist es nur so möglich, dass große Softwareunternehmen in kurzer Zeit Reichtümer anhäufen, die in der „alten" Ökonomie unvorstellbar wären. Sobald der Absatz eines Produktes seine Fixkosten gedeckt hat, wird der weitere Umsatz fast zu Reingewinn. Andererseits zeigen Phänomene wie das Austauschen von Kopien (ob nun im Bekanntenkreis oder institutionalisiert an Tauschbörsen) deutlich, dass der Nutzer geistiges Eigentum anders begreift als materielles und nicht bereit ist, in der digitalen Welt der alten Verwertungslogik zu folgen.

II. Der Austausch von Wissen

Neben der kommerziellen, proprietären Verwertungslogik von Informationsgütern entwickelt sich seit geraumer Zeit eine andere Form der Herstellung und Verbreitung von Computerprogrammen und inzwischen auch von anderen Arten digitaler Güter. *Quelloffene Software*[2] scheint auf den ersten Blick einen besser an die Bedingungen der digitalen Welt angepassten Umgang mit geistigem Eigentum zu versuchen. Für den Nutzer ist sie oft kostenlos oder vergleichsweise kostengünstig, womit dem fast unendlichen Angebot Rechnung getragen wird. In der Konzeption von Free Software wird geistiges Eigentum an sich abgelehnt.

Doch wie lässt sich das Phänomen quelloffener Software charakterisieren? Die vorangegangenen Kapitel dieses Buches verdeutlichen, dass es Elemente eines neuartigen Vermarktungskonzepts und eines technischen Entwicklungsmodells enthält. Liest man jedoch die Verlautbarungen seiner Protagonisten, scheint die gesellschaftliche, vielleicht sogar gesellschaftsverändernde Komponente ebenfalls ein wichtiges Element zu sein.

Es erschien daher unumgänglich, in diesem Jahrbuch, das ja einen möglichst umfassenden Überblick über quelloffene Software geben soll, der gesellschaftlichen Komponente ihren Platz einzuräumen und auch die Geisteswissenschaften zu diesem technischen Problem zu Wort kommen zu lassen.

Die wissenschaftliche Erforschung quelloffener Software liegt zunächst schon deshalb auf der Hand, weil sie ja selbst aus der Wissenschaft kommt, ursprünglich sogar Wissenschaft war (hierzu und zum Folgenden vgl. Grassmuck 2002). Ihren Ursprung nahm sie Anfang der achtziger Jahre an der Berkley-Universität mit der Weiterentwicklung des Betriebssystems UNIX und dessen freier Veröffentlichung unter der Lizenz *Berkeley Software Distribution* (BSD). Die Veröffentlichung war keineswegs durch Altruismus motiviert, sondern Teil des normalen, jahrtausendealten wissenschaftlichen Prinzips des *peer review*. Andere mit der Materie vertraute Informatiker konnten so ihre Kommentare abgeben, eigene Ideen entwickeln und das Projekt UNIX weitertreiben.

[2] Wenn hier von quelloffener Software die Rede ist, sind vereinfachend beide Ausprägungen dieses Phänomens, „Open Source" und „Free Software", gemeint. Die Unterschiede sind in den vorangegangenen Kapiteln ausführlich dargestellt.

Der Rest der Geschichte ist an anderer Stelle dieses Buches erschöpfend dargestellt: Die Entwicklung des Internets vergrößerte den Kreis der Entwickler weit über die Universitäten hinaus, Richard Stallman, ein Programmierer des MIT, gründete die *Free Software Foundation* (FSF), um Software vor der zunehmenden Kommerzialisierung und den damit einhergehenden Einschränkungen durch *Copyrights* zu bewahren. Die FSF entwickelte eine Lizenz, die die Prinzipien von *Free Software* schützen sollte,[3] und aus wissenschaftlichen Projekten wurden für einen breiten Nutzerkreis geeignete Programme wie das UNIX-Betriebssystem *GNU/Linux*. Von der Free-Software-Gemeinde spaltete sich die Open-Source-Bewegung ab, weitere Projekte wurden verwirklicht, und schließlich erfreuen sich heute die Produkte quelloffener Software bei gewerblichen wie privaten Anwendern zunehmender Beliebtheit, nicht nur, weil sie oft kostenlos oder sehr kostengünstig angeschafft werden können, sondern vor allem, weil Sicherheitslücken und Fehler auf Grund des Prinzips des *peer review* in einer riesigen virtuellen Entwickler- und Testergemeinde oft schneller beseitigt werden als bei den meist sehr teuren proprietären Konkurrenzprodukten.

Grassmuck (2002) vergleicht die Entwicklung des Wissens mit dem Schicksal der Allmende. Diese Form des allgemeinen Nutzungsrechts – mit Eigentum hat sie zunächst wenig zu tun – war noch bis ins 19. Jahrhundert in Europa weit verbreitet. So konnten die Mitglieder beispielsweise eines Dorfes auf der Gemeindewiese ihr Vieh weiden, in einem ausgewiesenen nahe gelegenen Wald Brenn- und Bauholz schlagen oder am gemeinsamen Brunnen Wasser schöpfen. Seit dem ausgehenden Mittelalter wurden nun die Allmenden nach und nach in Privatbesitz überführt, die Möglichkeit ihrer kostenfreien Nutzung ging in der Folge verloren. Diese Entwicklung wird allgemein dadurch gerechtfertigt, dass die klassischen Allmenden wegen ihrer Erschöpfbarkeit und der fehlenden Verantwortung des einzelnen Nutzers nicht nachhaltig bewirtschaftet werden können.

In ähnlicher Weise versteht Grassmuck (2002) den Zugang zu Wissen, der sich spätestens seit der Zeit des Humanismus verbreiterte. Durch den verbesserten Buchdruck wurde „Wissen" für größere Bevölkerungsschichten erschwinglich, öffentliche Leihbibliotheken, Schulen, Universitäten etc. kamen hinzu. Lange Zeit wäre niemand auf die Idee gekommen, dass der Inhalt von Büchern einen eigenen materiellen Gegenwert besitzen könnte, dass eine Idee Eigentum sein kann. So wie die Gemeindewiese dem Dorfbewohner zum Weiden seiner Schafe zur Verfügung stand, sollte das Wissen dem Untertanen zugänglich sein. Erst im Zusammenhang mit der Industrialisierung wurde begonnen, diese unendliche Wiese der Erkenntnis einzuzäunen. Zunächst waren es Patente, die „Wissen" zu Privateigentum werden ließen, im 20. Jahrhundert entstand dann ein umfangreiches System des *intellectual property*. Nur die Wissenschaften (die anders auch gar nicht möglich wären) haben sich den Allmende-Gedanken erhalten, nur hier ist es möglich geblieben, die Ideen eines anderen zu verwenden, solange man nur seine Autorschaft kenntlich macht. Würde aber der Zugang zu Informationen nicht durch künstliche Zäune auf der Wiese des Wissens beschränkt, dann entstünde eine unendliche Wissens-Allmende,

[3] Die GNU *General Public License* (GPL), abrufbar unter http://www.gnu.org/licenses/licenses.html.

die unter dem klassischen Problem der Gemeindewiese nicht leiden würde: Sie kann durch die vielen Schafe, die sie betreten, nie zerstört werden.

Richard Stallman, der Begründer der FSF und bekannteste Vordenker des Modells der quelloffenen Software, vertritt vehement die Freiheit von Wissen, speziell von Software. Er betrachtet die gängigen *Copyright*-Regeln als regelrechten Verrat:

> Signing a typical software license agreement means betraying your neighbor: ‚I promise to deprive my neighbor of this program so that I can have a copy for myself.' People who make such choices feel internal psychological pressure to justify them, by downgrading the importance of helping one's neighbors – thus public spirit suffers (Stallman 1992)

Im Fazit wird dann deutlich, dass er im geistigen Eigentum ein gesellschaftliches Problem sieht:

> Software hoarding is one form of our general willingness to disregard the welfare of society for personal gain. [...] We can measure it with the size of the homeless population and the prison population. The antisocial spirit feeds on itself, because the more we see that other people will not help us, the more it seems futile to help them. Thus society decays into a jungle.
>
> If we don't want to live in a jungle, we must change our attitudes. We must start sending the message that a good citizen is one who cooperates when appropriate, not one who is successful at taking from others. I hope that the free software movement will contribute to this: at least in one area, we will replace the jungle with a more efficient system which encourages and runs on voluntary cooperation (ebd.).

Freie Software ist für ihren Erfinder (oder „Wiederentdecker") also längst kein Selbstzweck, sie ist Teil einer gesellschaftlichen Utopie, derzufolge Information für jeden frei zugänglich sein soll. Die Beschränkungen am Zugang zu geistigem Eigentum werden hier Teil eines größeren Problems, das die Gesellschaft in den Abgrund zu stoßen droht. Freie Software könnte einen Ausweg darstellen.

Eric S. Raymond widerspricht dieser Auffassung. Er wendet am Beispiel von Linux ein, dass Programmierer in einem Open-Source-Projekt oft Pragmatiker sind, die ihre Arbeit nicht wegen des hehren Prinzips der Freiheit (von Informationen) verrichten, sondern schlicht, weil sie Spaß am Programmieren haben und gute Software herstellen wollen (Raymond 2000). Die Möglichkeit geistigen Eigentums auch an Open-Source-Produkten lässt er ausdrücklich zu, die Kriterien für Open-Source-Software[4] dürfen dabei aber nicht verletzt werden (ebd.). Doch auch Raymond findet seine gesellschaftliche Utopie. Wenn die klassische Ökonomie in einer Gesellschaft des Tausches stattfindet, in der der Einzelne soziales Prestige durch die Kontrolle über möglichst viele (materielle) Güter erreicht, so siedelt er die *community* der quelloffenen Software dagegen in einer Welt der Geschenk-Kultur an, in der soziales Prestige durch Geschenke erworben wird. Als Vorbild dienen ihm die Kwakiutl-Indianer und der ausgeprägte Hang zur Wohltätigkeit bei Multimillionären.

[4] Wie Weitergabe des Quellcodes, Kenntlichmachung von Änderungen etc. Die Prinzipien von Open Source sind an anderer Stelle des Buches ausführlich dargestellt und unter http://www.opensource.org/docs/definition.php einzusehen.

Weil nun die Protagonisten quelloffener Software selbst in ihrem Entwicklungsmodell oft etwas Neues, Gesellschaftsveränderndes, ja teilweise Antikapitalistisches sehen, liegt eine Beschäftigung mit dem Thema unter dem Blickwinkel der Geisteswissenschaften nahe. Schon bei oberflächlicher Betrachtung von „Open Source" und „Free Software" werfen sich Fragen auf. Haben wir es mit einer sozialen Bewegung von Kritikern der Informationsgesellschaft zu tun, ähnlich der Umweltbewegung in der Industriegesellschaft? Oder sind jene Programmierer, die sich in riesigen Gruppen virtuell zusammenschließen, um informatische Probleme zu lösen, bunte Vögel, Hobbybastler und langfristig doch eine Randerscheinung? Stehen hinter diesen Menschen Ideen, ja vielleicht ganze philosophische Konzepte, die den Geisteswissenschaften altbekannt sind? Diese Fragen lassen sich derzeit nicht erschöpfend beantworten, weil das Phänomen der quelloffenen Software selbst noch zu jung ist, ihr weiteres Schicksal im Dunkeln liegt und die Erforschung desselben eben erst begonnen hat. Veröffentlichungen zu diesem Thema sind oft von ideologischen Vorbehalten geprägt, nüchterne Analysen sind noch selten und teilweise aus begeisterter bzw. ablehnender Polemik schwer herauszulesen. Dieses Kapitel will einen Einblick in den Forschungsstand der Geistes- und Sozialwissenschaften geben.

Die Soziologie beginnt gerade, den gesellschaftlichen Kontext von quelloffener Software zu erforschen. Hier untersucht der Aufsatz „Heterogene Ingenieure. Open Source und Freie Software zwischen technischer und sozialer Innovation" von *Ursula Holtgrewe* die soziologischen Hintergründe der Open-Source-Welt mit der Fragestellung, ob die soziale Basis der Entwicklergruppen das oft geäußerte Ideal rechtfertigt, die Prinzipien dieser offenen, freien und selbstbestimmten Arbeits- und Wirtschaftsweise könnten auch in anderen gesellschaftlichen Sphären Anwendung finden.

Thomas Zimmermann geht in dem Beitrag „Open Source und Freie Software – soziale Bewegung im virtuellen Raum?" der grundsätzlichen – und längst nicht entschiedenen – Frage nach, ob es sich bei den Gruppen, die sich virtuell um Open-Source- und Free-Software-Projekte sammeln, um soziale Bewegungen im Sinne der Arbeiterbewegung des 19. und der Umweltbewegung des 20. Jahrhunderts handelt.

Die Gedankengebäude und die Ideengeschichte, die implizit und explizit hinter quelloffener Software und den sich um sie bildenden Gemeinschaften stehen, werden von der Philosophie erforscht. *Karsten Weber* wird dies hier in dem Aufsatz „Philosophische Grundlagen und mögliche Entwicklungen der Open-Source- und Free-Software-Bewegung" tun, indem er bekannte theoretische Denkmodelle in diesem neuen Zusammenhang untersucht. Dabei kommt er zu der erstaunlichen Entdeckung, dass sich in der „Open-Source-Philosophie" libertäre und kommunitaristische Positionen, die sonst in starkem Gegensatz zueinander stehen, vereinen. Auf dieser Basis versucht er einen vorsichtigen Blick in die Zukunft.

Open Source und Free Software werden immer wieder als Gegenentwurf zum Kapitalismus verstanden. Die Ideengebäude der Theoretiker und Entwickler quelloffener Software unter dieser Fragestellung zu untersuchen, haben sich *Lydia Heller und Sabine Nuss* in ihrem Beitrag „Open Source im Kapitalismus: Gute Idee – falsches System?" zur Aufgabe gemacht. Während sie analysieren, inwieweit sich dieses Phänomen vielleicht doch innerhalb der alten Verwertungslogik entwi-

ckelt, richten sie ein besonderes Augenmerk auf die gängigen Theorien zum geistigen Eigentum.

Uli Zappe geht zunächst bis ins 18. Jahrhundert zurück, um in der Moralphilosophie eine Antwort auf die Frage zu finden, ob nicht-knappe Güter weitergegeben werden müssen oder ob sie ebenso am Markt gehandelt werden können wie klassische knappe Güter. Auf dieser Grundlage verfolgt er die Wurzeln nicht-konservativer Modernisierungskritik und schlägt vor, quelloffene Software im Sinne Friedrich Schillers als Gegenentwurf zur rein ökonomischen Rationalisierung der Aufklärung zu verstehen.

Auf Grund des beschränkten Rahmens dieses Buches müssen trotzdem viele Fragen offen und viele Theorien unerwähnt bleiben. Jedoch: Ein kleines Steinchen zum Verständnis dieses – eben nicht nur technischen – Phänomens kann dieses Kapitel hinzufügen. Einiges ist nur verschoben auf die folgenden Jahrbücher, wie bspw. der Beitrag von Richard Stallman, der auf Grund einer Erkrankung des Autors nicht zustande kommen konnte. Anderes wird sich durch die hier geäußerten Gedanken erst noch ergeben.

Literatur

Grassmuck, Volker (2002): *Freie Software. Zwischen Privat- und Gemeineigentum*, Bonn 2002,
online http://freie-software.bpb.de (20.12.2003).

Stallman, Richard (1992): *Why Software Should Be Free?* In: Free Software, Free Society. Selected Essays of Richard M. Stallman, Boston 2002,
online http://www.gnu.org/philosophy/shouldbefree.html (15.1.2004).

Raymond, Eric S. (2000): *Homesteading the Noosphere*,
online http://www.catb.org/~esr/writings/homesteading/homesteading/index.html (15.1.2004).

Zappe, Uli (2003): *Das Ende des Konsums in der Informationsgesellschaft*, in: Fischer, Peter / Hubig, Christoph / Koslowski, Peter (Hg.): Wirtschaftsethische Fragen der E-Economy, Heidelberg 2003, S. 48 ff.
online http://www.ritual.org/Herbst/Konsumende.pdf (29.9.2003).

Heterogene Ingenieure – Open Source und Freie Software zwischen technischer und sozialer Innovation

URSULA HOLTGREWE

1. Open Source als Innovations- und Gesellschaftsmodell?

Die Entwicklung freier und Open-Source-Software[1] ist in letzter Zeit nicht mehr nur unter IT-ExpertInnen auf Interesse gestoßen. Innovationsforscher sehen hier einen Weg, der technische und soziale Innovation verbindet und dabei einige der Dilemmata und Risiken der Innovation für den Markt auflöst (Lakhani und von Hippel 2000; Moon und Sproull 2000; Tuomi 2002). GesellschaftstheoretikerInnen sehen im Entwicklungsmodell freier Software gar ein Modell für die Produktions- und Lebensweise von Wissensgesellschaften, das, je nach Standpunkt, die Institutionen kapitalistischer Produktion überwinden könnte (Grassmuck 2000; Meretz 2000; Himanen 2001; Gorz 2002).

So attraktiv also das Modell FS/OS innovations- und gesellschaftstheoretisch ist, stellt sich doch die Frage seiner Reichweite und Verallgemeinerbarkeit. Wie sehen die Voraussetzungen und „Erfolgsgeheimnisse" der freien Software empirisch aus, wenn man sie auf die Struktur von Projekten und die Arbeitsweisen und Motivationen der EntwicklerInnen bezieht? Die These dieses Beitrags ist, dass die virtuelle, offene und selbstbestimmte Kooperation, die das Entwicklungsmodell FS/OS ausmacht, auf sehr spezifischen sozialen und technischen Voraussetzungen beruht, die einander wechselseitig stützen. Ob die proklamierte Offenheit und die Partizipationschancen in der Tat zu einer Demokratisierung von Technikentwicklung führen oder ein letztlich exklusives Modell darstellen, ist sozial noch nicht entschieden.

1.1 Innovation im Netz

Innovationen sind in Organisationen und auf Märkten grundsätzlich so notwendig wie problematisch. Schon die Erzeugung neuen Wissens und dessen Überführung in Innovationen erzeugt Unsicherheit: Wer mit welchem Wissen und mit wie viel Aufwand in welcher Zeit eine Erfindung hervorbringt, die sich dann auch als Innovation verwerten lässt – das kann ein Unternehmen im Vorfeld nicht wissen, denn sonst wäre es keine Innovation (Rammert 1988; Ortmann 1995). Sodann stellen Innovationen – abhängig von ihrer Reichweite – gerade die Stärken bisheriger Organisationsprozesse und Strukturbildungen in Frage. Regeln und Routinen stehen zur Disposition, Ressourcen werden entwertet (Schumpeters „schöpferische Zerstörung", vgl. Rammert 1997). Umgekehrt aber setzt die langfristige Bestandserhaltung

[1] Die Rede von der freien Software hat bekanntlich eher die Assoziationen sozialer Bewegung, die von Open Source bezieht sich auf den offenen Quellcode und die technische Exzellenz. Da die Debatte aus guten Gründen nicht zu entscheiden ist, verwende ich das Kürzel FS/OS.

von Organisationen – zunehmend – deren Fähigkeit voraus, Innovationen hervorzubringen und Innovativität zu erhalten. Sie müssen die damit verbundenen Risiken also übernehmen und gleichzeitig versuchen, sie in Grenzen zu halten oder zu externalisieren – um so mehr dann, wenn es um die Sicherung der Innovationsfähigkeit auf Dauer geht.

Auch Märkte aber bringen Innovationen keineswegs unproblematisch hervor, weil und wenn Innovationen neues Wissen enthalten (Beckert 1997). Über die Zukunft liefert der Markt keine Informationen, und den Wert des Wissens können Unternehmen weder exakt bestimmen noch vollständig aneignen. Wissen ist nicht teilbar, man kann es gleichzeitig behalten, verkaufen und verschenken. Andere davon auszuschließen, ist nur über die materielle Beschaffenheit von Wissensgütern und/oder über komplexe rechtliche und technische Vorkehrungen möglich. Der gesamtgesellschaftliche technische Fortschritt, der aus der Zirkulation von neuem Wissen entsteht, ist aus der Sicht der Verwertung einzelner Kapitalien entgangener individueller Profit.

Die Attraktivität des FS/OS-Innovationsmodells liegt nun darin, technische Innovation zunächst jenseits von Unternehmen und Märkten hervorzubringen und damit deren Versagensanfälligkeit zu umgehen. Es macht deutlich, dass funktions- und konkurrenzfähige Technik jenseits des Marktes entwickelt werden kann – und dies in global vernetzter, freiwilliger Kooperation, in der Qualität gesichert und Wissen ausgetauscht wird. Offensichtlich aber lässt sich dieses Entwicklungsmodell auch wieder an die Strategien von Unternehmen und das Agieren auf Märkten anschließen. FS/OS-Dienstleister etwa entwickeln Geschäftsmodelle auf der Basis öffentlicher Güter, indem sie ihr Geld mit Distribution oder kundenspezifischer Anpassung verdienen. Für Hard- und Softwarehersteller stellt die Unterstützung von FS/OS einen Weg dar, zu niedrigen Kosten Alternativen zur „Umklammerung" durch den Monopolisten Microsoft zu erschließen und Handlungsspielräume auszuweiten. Die Lizenzen machen dabei die Bedingungen und Risiken der Vermarktung kalkulierbar: Wenn ein Unternehmen Entwicklungen nicht privatisieren kann, so können seine Wettbewerber das auch nicht (Holtgrewe und Werle 2001; Winzerling 2002).

Insofern fügt sich das Innovationsmodell FS/OS gut in die Befunde der neueren Innovations- und Technikforschung, die immer wieder die Bedeutung von Netzwerken unterschiedlicher Akteure und von institutionellen „Rahmen"-Bedingungen für Innovationen hervorhebt. Technische Innovationen bewegen sich im Prozess ihrer Entstehung, Stabilisierung und Durchsetzung durch verschiedenartige Handlungssphären und Netzwerke, und der Aufbau solcher Netzwerke ist Teil innovativen Handelns (Rammert 1997; Weyer u.a. 1997; Sauer und Lang 1999). Gerade aus der Kombination und Artikulation von Ressourcen und Regeln, Deutungen und Normen heterogener Akteure entsteht Neues. Die Deutung und Bewertung des Neuen ist wiederum eingelassen in Innovationsmilieus und -paradigmen. Sie ist etabliert in technischen Leitbildern oder den „mitgebrachten" professionellen und kulturellen Orientierungen der Innovatoren, aber diese Kulturen und Identitäten werden im selben Prozess reproduziert und transformiert.

Die marktliche und betriebswirtschaftliche Logik ist also in Innovationsprozessen generell eingebettet in andere, technische, ästhetische oder professionelle Orientierungen. Bekanntlich arbeiten Wissenschaftlerinnen, Künstler, Erfinderinnen und Unternehmer nur zum Teil für Geld. Spaß an der Sache, Lust an den eigenen Fähigkeiten, der „Traum, ein privates Reich zu gründen" (Schumpeter), die Reputation, die man als UrheberIn von Innovationen erwirbt, und vielleicht auch ein ganz altruistisches Interesse, die Welt zu verbessern und zu verschönern, motivieren InnovatorInnen. Im Kontext kommerzieller Innovation sind diese Motive gewissermaßen privat – eine Sache individueller Marotten und oder kultureller Restbestände, aber ohne diese geht es nicht. Erfolgreiche Innovatoren sind aus diesen Gründen „heterogene IngenieurInnen" (Law 1987), denen es gelingt, nicht nur Technologien, sondern Akteure, Werkzeuge, Deutungen und Normen zusammenzufügen – aber die Nutzung der Innovation bringt diese wieder in neue Kontexte, in denen das Aneignen, Umdeuten, Ressourcen-Mobilisieren und Umbauen weitergehen.

„Networks of innovation" (Tuomi 2002) und heterogene Akteure und Motive sind also nicht FS/OS-spezifisch. Spezifisch ist jedoch das Maß an sozialer Phantasie und Reflexivität, das diese heterogenen Ingenieure auf die sozialen Voraussetzungen der Herstellung von Wissen und Technik als öffentliche Güter richten. Sie entwickeln nicht nur die technischen, sondern auch die rechtlichen und institutionellen „Werkzeuge" der Kultivierung selbstbestimmter Kreativität. Die GPL[2] etwa („All rights reversed") nutzt das (us-amerikanische) Urheberrecht, um die weiterhin produktive und kreative Nachwelt an die Bedingungen offener Nutzung zu binden – eine Selbstbindung (Elster 1987) an Freiheit, und ein Versuch, sich und die Nachwelt auf Vielfalt festzulegen. An dieser Stelle, wo die Bedingungen kreativen Handelns reflektiert und ausgeweitet werden und man dazu die sozialen Institutionen der Innovation kritisiert und umbaut, sehe ich die Dimension sozialer Bewegung.

1.2 „Betriebssystem einer freiheitlichen Gesellschaft?" [3]

In der selbstbestimmten, global verteilten Kooperation sehen AutorInnen aus den Sozial- und Kommunikationswissenschaften ein weit reichendes Gesellschaftsmodell für die Wissens- und Informationsgesellschaft. André Gorz etwa sieht hier im Anschluss an die Marxsche Sicht aus den „Grundrissen" (Marx 1974), den „Ansatz zu post-kapitalistischen Produktionsverhältnissen", in welchen Gebrauchswerte nicht mehr als Waren erzeugt und getauscht werden: in denen Wissen als Gemeingut aller Menschen allen zugänglich ist; in denen mit der Warenbeziehung auch der Konkurrenzkampf und der Zwang zur Leistungsmaximierung aufgehoben ist; in denen die Qualität der Gebrauchswerte, der Genüsse, der Bedürfnisbefriedigung und der sozialen Beziehungen zum entscheidenden Ziel erhoben" werden kann (Gorz 2002, S. 32).

Die Verbindung von Gebrauchswert, Bedürfnisbefriedigung und sozialen Beziehungen stellt der Open-Source-Programmatiker Eric Raymond mit einem merklich anderen, eher us-amerikanisch-libertären Akzent her: Der Geschenk- und Reputa-

[2] Die GNU *General Public License* der Free Software Foundation ist einzusehen auf http://www.gnu.org/copyleft/gpl.html.

[3] Vgl. Grassmuck 2000.

tionsökonomie, die Raymond beschreibt (1998; 1999a) liegt ein Modell autarker ProduzentInnen zu Grunde, die sich als Siedlergemeinschaft in den Sphären des Immateriellen die eigenen Werkzeuge selbst herstellen. Hier wird ein Stück weit nostalgisch eine Ganzheitlichkeit des Produzierens beschworen, die quasi handwerklich gedacht ist. Die Autarkie der Einzelnen hat zwar ihre Grenze, weil nicht jede alles können kann und muss, und gerade die Gemeinschaft effizienter produziert und Fehler korrigiert, aber Raymonds Gesellschaftsmodell fokussiert gut-amerikanisch individualistisch auf die Kreativität vieler Einzelner stärker als den sozialen Zusammenhang.

Ilkka Tuomi (2002) hingegen macht – von der Innovationsforschung herkommend – in der Hackerkultur eine genuin moderne Synthese technischer und kommunikativer Rationalität aus, die man ja in der Tradition der Kritischen Theorie als eher gegensätzlich betrachten würde. Hier steht der „eigentümlich zwanglose Zwang des besseren Arguments" (Habermas 1971, S. 137) nicht entgegen, sondern im Einklang mit dem technischen Funktionieren: „The culture of hacking is probably the most perfect and frictionless implementation of modernity. [...] As long as it builds itself around those technological artefacts that it produces, it is able to avoid many of those conflicts that make similar efficiency difficult in broader social contexts" (Tuomi 2002, S. 214). Damit freilich weist er schon auf die möglichen Grenzen des FS/OS-Innovationsmodells als Gesellschaftsmodell hin. „Running code" ist ja eine hoch spezifische Basis für einen hinreichenden Konsens, und eine geteilte Orientierung auf „die Sache" kann schwerlich in allen gesellschaftlichen Sphären als Verständigungsgrundlage dienen.

Vielleicht ist es an dieser Stelle sinnvoll, wieder an die Einsichten kritischer Theorie und feministischer Rationalitätskritik (Smith 1987; Benhabib 1992; Fraser 1994) zu erinnern: nämlich, dass sowohl die technische als auch die kommunikative Rationalität mit ihren Unterstellungen umfassend kompetenter und unabhängiger Subjekte konstitutiv auf bestimmte Ausschließungen und Ausblendungen angewiesen sind. Zu den grundlegenden Problematiken etwa von Bedürftigkeits- und Sorgebeziehungen zwischen Menschen und damit zur Frage, wo denn die technisch und kommunikativ kompetenten Individuen herkommen, kann weder die Beschwörung von virtuellen Pioniergemeinschaften noch die „reibungslose Implementation der Moderne" sehr viel beitragen, und eben dies macht ihre Effizienz aus. Das bedeutet nicht, dass jede soziale Bewegung zu jedem sozialen Problem einen „Standpunkt" haben müsste, aber es sollte die Promotoren der verschiedenen „Hackerethiken" zu Bedacht auf die eigenen Voraussetzungen und Ausblendungen anregen.

Das „Erfolgsgeheimnis" der FS/OS also liegt – so die These dieses Beitrags – weniger in ihrem gesamtgesellschaftlichen Modellcharakter als in ihrer Besonderheit und Selektivität einerseits, ihrer Integration von technischen und sozialen Innovationen andererseits – und die Selektivität des thematischen Fokus ist es, die diese Integration erst ermöglicht. Technikentwicklung selbstbestimmt über eine technisch und sozial lockere Kopplung von Projekten und Kooperationen zu betreiben und die sozialen und institutionellen Innovationen zu entwickeln, die dieses Model stabilisieren, ist sozial hoch voraussetzungsvolles *heterogeneous engineering*.

2. Aktivisten und Ingenieure?

Auch innerhalb der FS/OS-*communities* sind die Bezüge auf technische und soziale Innovation, auf technische Exzellenz und gesellschaftliche Transformation vielfältig. Manuel Castells, Theoretiker der „Netzwerkgesellschaft" (Castells 1996) hat skizziert, wie bereits bei der Entwicklung des Internet die professionellen und akademischen Orientierungen auf technische Exzellenz und kollegiale Kooperation mit den Überzeugungen sozialer und gegenkultureller Bewegungen von freier und kreativer Entfaltung ineinandergriffen (Castells 2001; Holtgrewe und Werle 2001). Mit der Entstehung freier und Open-Source-Software differenzierten sich in der Diskussion um Lizenzen und politische Positionierung diese Positionen alsbald aus, und es ließen sich „Aktivisten" im Sinne sozialer Bewegung und „Ingenieure" im Sinne technischer Exzellenz unterscheiden (Holtgrewe 2001). Insistiert etwa Richard M. Stallman auf dem weit gehenden sozialen und philosophischen Anspruch der Freiheit, die vom Gebrauchswert, der kreativen und intelligenten Nutzung und Weiterentwicklung der Produkte her gedacht wird, so hielt Eric S. Raymond solche „philosophischen" Ansprüche und Programme für eher reputationsschädlich im Kontext industrieller Anwendung: „Shut up and show them the code!" (Raymond 1999b).

Dabei haben die Debatten um Lizenzen und Selbstverständnis zwar immer wieder Konflikte, Spannungen und Parallelentwicklungen hervorgerufen, aber es gelingt den *communities*, sich nicht in Flügel- und Sektenkämpfen aufzureiben. Einflussreiche Akteure, wie etwa Linus Torvalds, vertreten öffentlich immer wieder pragmatisch-undogmatische Positionen und treffen entsprechende Entscheidungen, und die GPL wird bislang auch von Unternehmen respektiert (O'Mahony 2003) – wiewohl abzuwarten bleibt, ob die Klage der Firma SCO gegen IBM die hier bislang etablierten Spielregeln in Frage stellt (Landley und Raymond 2003). In Auseinandersetzungen um nicht- oder teilweise freie Lizenzen wie im Falle von KDE, haben Unternehmen sich vielfach auch von der GPL oder einer anderen „freieren" Lizenz überzeugen lassen (Moody 2002).[4]

Das hat selbstverständlich nicht nur normative und politische Gründe. Auch den Unternehmen ermöglichen FS/OS-Projekte ja die Erschließung von Innovationspotenzialen über die eigene Organisation hinaus, die das Innovationsrisiko verringert. Bei freiwilliger Tätigkeit müssen sie den Einsatz von Zeit und Arbeit mit ungewissem Ergebnis nicht erst als solches kalkulieren oder nur einen Teil der Innovationskosten aufbringen. Begeben sich Unternehmen in dieses Feld, so lassen sie sich damit auf die neuen Spielregeln ein (O'Mahony 2003): Zum einen aus guten ökonomischen Gründen, zum anderen aber auch unter der Perspektive, einen fairen sozialen Tausch mit der *non-profit-community* einzuhalten. Ob man nun die GPL für die

[4] Eine technische Lösung für das soziale Problem einer mehr oder weniger puristischen Position hat die nichtkommerzielle Linuxdistribution *debian* entwickelt: Hier gibt es einen „virtuellen Richard M. Stallman" (http://packages.debian.org/unstable/admin/vrms.html), ein Programm, das die aktuelle *debian*-Installation auf nicht GPL-lizenzierte Bestandteile überprüft (die bei *debian* in einem separaten Verzeichnisbaum liegen). In Zukunft soll das Programm zudem auf Wunsch Auszüge aus Stallmans Veröffentlichungen anbieten, die die moralischen Probleme mit nicht-GPL-Lizenzen erläutern.

einzig mögliche Lizenz hält oder nicht: Mit ihrer Selbstbindungswirkung hat sie nicht nur die Spielregeln der Innovation verändert und die Möglichkeiten selbstbestimmter Kreativität erweitert, sondern auch die Frage der Nachhaltigkeit und Kontinuität der Produktion öffentlicher Güter auf die Tagesordnung gesetzt.

3. Die Empirie: Wer macht was warum?

Wie weit aber nun die guten Gründe ökonomischer und normativer Art, die für FS/OS sprechen, sich in der Praxis wiederfinden und EntwicklerInnen motivieren, ist die Frage. Die emphatische Beschwörung der Hackerkultur (Himanen 2001), der kreativen Selbstentfaltung oder gar einer pionierhaften Siedlergemeinschaft in den Sphären des Immateriellen (Raymond 1999a) sind sozial wenig aussagekräftig. Journalistische (Moody 2002) und biographische Berichte über zentrale Personen (www.softpanorama.org) sind hochinteressant, aber bleiben dennoch bei einer Art Geschichte großer Männer stehen.

Das allein ist schon aufschlussreich. Es scheint, dass die Projekte mit vergleichsweise wenig Informationen über ihre Mitglieder auskommen, sich also auf den ersten Blick ein Stück weit indifferent gegen deren konkrete Motivation, Herkunft, formale Qualifikation usw. setzen:

> Libre Software applications have been able to prove their quality without any knowledge of the people dedicated to producing it. [...] When a developer wants to participate in a Libre Software project, nobody will ask him neither for his nationality, nor for his profession nor for his age. If he wants to contribute in a constructive way and has the necessary knowledge or skills, yet even if he is only just on the way to acquire them, he is invited to do so. This is said to point out that Libre Software development has been carried out without considering the aspects this study is all about – the personal information. That is good, and it has to remain as is. (Robles u.a. 2001, S. 2).

Das unterstützt Tuomis These der effizienten und auf den Punkt gebrachten Implementation der Moderne: Bestand die Effizienzsteigerung der klassisch-bürokratischen Organisation nach Max Weber (1980) zumindest im Modell darin, „ohne Ansehen der Person" zu operieren und ihre Mitglieder nicht nach der Herkunft, sondern nach der Qualifikation zu rekrutieren, so geht es nunmehr nicht um die Besetzung von Positionen in Organisationen, sondern um Beiträge „pur", die der sachlich-fachlichen Bewertung durch die *peers* unterzogen werden. Ebenso aber, wie sich auf den zweiten Blick die Posten in bürokratischen Organisationen durchaus nicht allein nach der Qualifikation, sondern nach Geschlecht, Alter, Ethnizität usw. recht unterschiedlich verteilen, so ist auch die Beteiligung an OS/FS-Projekten bei aller proklamierten Offenheit an bestimmte soziale und subjektive Voraussetzungen gebunden, die sich bei bestimmten sozialen Gruppen, jüngeren männlichen Hochqualifizierten, konzentrieren.

In den letzten Jahren sind einige Befragungen von OS/FS-EntwicklerInnen durchgeführt worden, die ein ziemlich konsistentes Bild ergeben (Robles u.a. 2001; Ghosh u.a. 2002; Lakhani u.a. 2002; Hertel u.a. 2003).[5] EntwicklerInnen sind jung,

[5] Lakhani u.a. haben 684 EntwicklerInnen befragt, die aus einer Zufallsauswahl von Projekten auf der

ihr Durchschnittsalter liegt unter 30 Jahren. Sie sind mit 98 – 99% fast ausschließlich männlich. 60 – 70% habe einen Hochschulabschluss, 20 – 30% studieren. Rund 80% arbeiten im IT-Bereich, die Mehrzahl als Programmierer oder Softwareentwickler, aber es gibt auch Wissenschaftler, Manager, Berater oder Administratoren. FS/OS-Entwicklung ist damit eine Sache von IT-Professionellen und Studierenden, aber ein rundes Fünftel „Amateure" ist für hochqualifizierte Entwicklungsarbeit auch nicht wenig.

Der zeitliche Einsatz der Befragten liegt für die Mehrheit im Rahmen eines zeitaufwändigen Hobbys. Ca. 2/3 (Robles u.a. 2001; Ghosh u.a. 2002) arbeiten weniger als zehn Stunden pro Woche an OS/FS. Nur eine Minderheit (5,6% bzw. 7,1%) verbringt damit mehr als 40 Wochenstunden. Bei Hertel u.a. (2003) lag der zeitliche Einsatz höher: bei durchschnittlich 18,4 Wochenstunden. Jedoch meinten 37,4% der von Lakhani u.a. (2002) Befragten, ihr zeitlicher Einsatz sei „completely variable". Interessant ist die empirische Verteilung bezahlter und unbezahlter Arbeit: Zwischen der Hälfte (Ghosh u.a. 2002) und 80% (Robles u.a. 2001) der Befragten arbeiten unbezahlt an FS/OS-Entwicklung. Im Sample von Hertel u.a. wurden 20% der Befragten vollständig für das Linux-Programmieren bezahlt, und weitere 23% verdienten gelegentlich Geld damit, was den höheren Zeitaufwand erklären mag.

Dabei gibt es Hinweise auf interessante Grauzonen zwischen Arbeit und Freizeit (Holtgrewe 2004): 12,8 % der von Ghosh u.a. Befragten entwickeln bei der Arbeit FS/OS, ohne dass sie direkt dafür bezahlt werden. Aus dem Sample von Lakhani u.a. sind es 17%, die bei der Arbeit entwickeln, ohne dass ihr Chef davon weiß. Für weitere 9% ist es nicht ihre Hauptaufgabe. Daraus lässt sich entnehmen, dass die FS/OS Projekte – neben der direkten Unterstützung durch Unternehmen – auch von den Spielräumen profitieren, die bestimmte IT-Fachleute bei der Arbeit haben. Hier finden sich gewissermaßen informelle Subventionen der FS/OS durch Arbeitszeit und *manpower*.

Das erinnert an überkommene Praktiken bestimmter Arbeiterkulturen, „während der Arbeit, für die man offiziell bezahlt wird, eigenen Beschäftigungen nachzugehen", wie sie der Praxistheoretiker Michel de Certeau skizziert hat (1988, S. 71): „Ein Arbeiter, der während der offiziellen Arbeitszeit für sich selber arbeitet,[...] entzieht der Fabrik Zeit, um frei, kreativ und vor allem nicht für den Profit zu arbeiten. Gerade an den Orten, welche von der Maschine, der er dienen muss, beherrscht werden, mauschelt er, um sich das Vergnügen zu verschaffen, zwecklose Produkte zu erfinden, die ausschließlich dazu dienen, durch sein ‚Werk' ein eigenes ‚Know-how' zum Ausdruck zu bringen" (ebda., vgl. Tuomi 2002, S. 28).

Open-Source-Plattform Sourceforge identifiziert und dann persönlich per Email angeschrieben wurden. Dabei ist mit einer Überrepräsentation us-amerikanischer EntwicklerInnen und der erfahreneren Betreiber eigener Projekte zu rechnen. Robles u.a. (2001, n = 5478) und Ghosh u.a. (2002, n = 2784) haben ihren Fragebogen ins Internet gestellt und die Einladung zur Teilnahme an eine Auswahl relevanter Mailinglisten und News-Foren geschickt. Die Antworten wurden über die E-Mailadresse mit den Autoren von OS-Code abgeglichen – signifikante Unterschiede zwischen den als EntwicklerInnen identifizierten und dem Rest des Sample ergaben sich nicht. Hertel u.a. (2003, n= 141, davon 69 aktive Entwickler) haben den Fragebogen ins Internet gestellt und über einschlägige Linux-Kernel-Mailinglisten dafür geworben.

Jedoch muss dies, wenn es sich um Wissensarbeit handelt, gerade kein illegitimes „Abzweigen" von Arbeit sein. Vielfach handelt es sich ja bei FS/OS-Entwicklungen um Werkzeuge für Softwareentwicklung und Administration, sodass die Arbeit an den Werkzeugen auch die „offiziellen" Jobs der EntwicklerInnen einfacher macht. Dann profitieren die FS/OS-Projekte vom schillernden Charakter von Informationsgütern und der Arbeit daran: Diese können gleichzeitig genutzt und verschenkt werden, und sie können dabei je unterschiedlich kontextuiert werden. Was für Kunden oder Arbeitgeber eine konkrete Problemlösung und Anpassung ist, wird nicht unbedingt weniger wertvoll, wenn man diese in ein FS/OS-Projekt einbringt. Gleichzeitigkeit und Mehrfachverwendung können hier zur Ressource werden. Wie konfliktfrei dies funktioniert, und wo Verwertungsinteressen von Arbeitgebern den EntwicklerInnen doch in die Quere kommen, wissen wir freilich nicht.

An Motiven für das Engagement in FS/OS-Projekten rangieren in den Befragungen intellektuelle Anregung, das Lernen, die Entwicklung und das Teilen der eigenen Fähigkeiten übereinstimmend oben. Programmieren macht den Befragten offensichtlich Spaß. Die Freiheit der Softwareentwicklung und das Kultivieren technischer Vielfalt werden durchaus als Werte vertreten. Normative und politische Überzeugungen, dass Code frei sein sollte (Lakhani u.a. 2002) oder Software nicht proprietär (Ghosh u.a. 2002), motivieren in den beiden Studien je ein Drittel der EntwicklerInnen. Lakhani u.a. leiten aus der Verteilung der Motive eine Typologie aus „learning and fun"-Orientierten (29%), „hobbyists" (27%), „professionals" (25%) und „community believers" (19%) ab.

Hertel u.a. (2003) vergleichen in ihrer sozialpsychologischen Studie die Motive der FS/OS-EntwicklerInnen zum einen mit Modellen für das Engagement in sozialen Bewegungen, zum anderen mit einem Modell der Motivation in virtuellen Arbeitsteams. Sie stellen fest, dass der zeitliche Einsatz – wie das Engagement in anderen sozialen Bewegungen – signifikant am stärksten mit der Identifikation der Befragten als Aktive, d. h. als Linux-EntwicklerInnen und EntwicklerInnen spezifischer Subsysteme zusammenhängt.[6] Einflussreich ist weiterhin die Erwartung persönlicher Vorteile und ein Faktor, den Hertel u.a. als „lack of concern for time losses" beschreiben. Gegenüber der Kooperation in anderen virtuellen Arbeitsteams spielt das Vertrauen, dass auch die anderen komplementäre Beiträge zum Ergebnis erbringen, eine nur geringe Rolle für das Engagement. In dieser Untersuchung sind die normativen Überzeugungen zwar präsent, aber sie wirken sich statistisch nicht auf den zeitlichen Einsatz und das von den Befragten in Zukunft erwartete Engagement aus.

Als wenig relevant für ihre Motivation betrachten die Befragten hingegen den Erwerb von Reputation (ca. 10% bei Ghosh u.a. 2002), wie es Eric Raymond (1998) nahe legt. Freilich funktioniert Reputation – wie im richtigen Leben – als Zustand, der nicht direkt, sondern wesentlich als Nebenfolge angestrebt werden kann (Elster 1987, S. 141 ff.). Zuzugeben, dass man Reputation direkt anstrebt, kann also den

[6] Das beantwortet, wie die Autoren selbst feststellen, nicht die Frage nach „Huhn oder Ei", ob die Identifikation das Engagement oder das Engagement die Identifikation hervorbringt. Mir scheint Letzteres wahrscheinlicher, wenn man annimmt, dass Menschen ihre Identität zentral durch praktisches Handeln entwickeln.

eher gegenteiligen Effekt haben, sodass das Maß, in dem FS/OS als Reputationsökonomie funktioniert, womöglich durch direkte Fragen nicht leicht zu erschließen ist.

Das Gesamtbild empirisch bei den EntwicklerInnen aufgefundener Motive lässt somit die Unterscheidung von „Ingenieuren" und „Aktivisten" als eine weitgehend programmatische erscheinen, die nach dem bisherigen Forschungsstand weder die Projekte noch die Fraktionen der Szene voneinander trennt. Sowohl pragmatische, technisch wie arbeitsmarktbezogen instrumentelle als auch normative und politische Motive spielen für die EntwicklerInnen eine Rolle Die reine Intrinsik aus Spaß am Programmieren und Freude am Entwickeln der eigenen Fähigkeiten rangiert in den meisten Befragungen weit oben – wenn man denn, wie die zitierten Befragungen dies tun, die möglichen Motive in eine Rangreihenfolge bringt. In eben dieser Intrinsik dürfte auch der Grund für die von Hertel u.a. bemerkte Toleranz für den zeitlichen Aufwand liegen. Wenn das Programmieren hinreichenden Spaß macht, muss es nicht unbedingt entlohnt oder durch Anreize motiviert werden. Dann sind Umwege, doppelte Anstrengungen oder Lösungen, die sich nicht verbreiten, nicht unbedingt ein Problem. Der Einsatz wird gerade nicht kalkuliert und die Zeit kann man nicht wirklich verschwenden. Das ist eine subjektive und lustvolle Weise, das Innovationsrisiko ungewisser Einsätze und Erfolge zu neutralisieren.[7]

4. Fazit: Soziale Bewegung und limitiertes Engagement

Gegenüber den idealisierenden Beschreibungen der „Hackerethik" oder der Pioniere im immateriellen Raum (Raymond 1999) nimmt sich das empirische Bild eher unspektakulär aus. Die hingebungsvollen Hacker-Idealisten, die jenseits alltäglicher Zeitbeschränkungen in enger Zwiesprache mit dem Rechner (auf Kosten menschlicher Kontakte) in virtuelle Räume aufbrechen, stellen schwerlich die Mehrheit der Szene. Aber auch das Bild der gehetzten SoftwerkerInnen, das die Industrie- und Arbeitssoziologie kennt (Beirne u.a. 1998; Glißmann 2002; Kalkowski und Mickler 2002), die sich in dichten Projekten mit entgrenzten Arbeitszeiten den Anforderungen und Zumutungen des Marktes nicht entziehen können, findet sich hier nicht wieder – was tautologisch ist: Wer keine Zeit dazu hat, wird sich auch nicht in FS/OS-Projekten engagieren. Wer OS/FS entwickelt, findet hingegen offensichtlich Zeit und Raum für selbstbestimmte Arbeit. Der Großteil davon liegt in der Freizeit, aber wir finden auch Hinweise auf eine Konversion von Erwerbsarbeit in öffentliche Güter, die mehr oder minder subversiv oder synergetisch ausfallen mag und die gerade die Begrenzungen spezialisierter und mitunter isolierter Problemlösung überwindet. Flexibilität und Grenzenlosigkeit der Arbeit und die begrenzte Aneigenbarkeit von Wissensproduktion bekommen hier einen eigenständig-produktiven Akzent.

[7] An diesem Motiv der Arbeit aus Leidenschaft, die kreative Leistungen und Innovationen hervorbringt und dazu Freiräume braucht und gestaltet, macht Himanen (2001) emphatisch die „Hackerethik" fest, die er – in Analogie zur Bedeutung der Weberschen protestantischen Ethik für die Entwicklung des Kapitalismus – als mögliches Wertsystem einer Wissensgesellschaft sieht.

Die soziale Offenheit der FS/OS-Entwicklung muss jedoch modifiziert werden. Sie beschränkt sich weitgehend auf die nahe liegenden „Verdächtigen": Jüngere männliche Hochqualifizierte. Auch wenn hier in *communities of practice* gelernt und Wissen ausgetauscht wird, qualifizieren die Projekte ihre Aktiven nicht von Anfang an. Vielmehr setzen diese ihre anderswo, vielfach an der Hochschule, erworbenen Programmier- und IT-Kenntnisse hier in einer informellen und selbstorganisierten Weise ein. Hier liegt eine Gemeinsamkeit zu den „Alternativ"-Projekten und Szenen der 80er Jahre, die ja vielfach von Studierenden auf der Suche nach anderen Formen des Arbeitens und Lernens getragen wurden. Im Unterschied zu deren Emphase auf „ganzheitlichen" Zusammenhängen von Arbeiten und Leben aber sind die Gegenstände der Kooperation eng fokussiert. Die Transparenz von Quellcode und Internet-Kommunikation erlaubt auf dieser Grundlage ein Lernen und Sich-Einarbeiten, das – im Unterschied zu den „klassischen" *communities of practice* (Lave und Wenger 1991) – in einer geringeren sozialen Dichte stattfindet. Bei aller Präsenz der Werte freier Kooperation und technischer Vielfalt im Feld ist es offensichtlich nicht zwingend, sich „ganz" zu einer *community* und deren Werten und sozial bewegten Ansprüchen zu bekennen – und ich vermute, dass diese mögliche Flexibilität und Selektivität des Engagements eher eine Stärke als eine Schwäche der Szene ist.

Im Lichte dieser Befunde lassen sich die nicht sehr geheimen „Erfolgsgeheimnisse" der FS/OS wie folgt skizzieren: Die Modularität von Programmen und Betriebssystem-Bestandteilen insbesondere in der Unix/Linux-Welt ermöglicht den kleinteiligen Charakter der Projekte (Ghosh und Prakash 2000). Die Transparenz von Quellcode und Internet-Kommunikation lässt es zu, dass kompetente Personen sich selbst rekrutieren (Benkler 2002) und Aufgaben selbstbestimmt zwischen Hobby und professioneller Arbeit definieren und übernehmen. Die Mehrfach-Verwendbarkeit von Wissensarbeit und Wissensgütern unterstützt dies.

Mit diesem technisch/sozialen Ensemble haben die FS/OS-EntwicklerInnen sich ein Handlungsfeld geschaffen, in dem viele der Risiken und Probleme organisierter und kommerzieller Innovationsprozesse neutralisiert sind: Der zeitliche Einsatz ist verteilt, selbstbestimmt und so weit intrinsisch motiviert, dass Verschwendung von Zeit für viele kein Problem ist – wiewohl sich die Projekte natürlich Werkzeuge zur Vermeidung von Doppelarbeit geschaffen haben. Technisch-soziale Festlegungen werden reversibel gehalten: „in open source, black boxes have transparent and penetrable walls" (Tuomi 2002, S. 7). Projekte können klein anfangen und sich und ihre Ambitionen entwickeln oder auch nicht. Indem man nicht für einen Absatzmarkt produziert, steht man gewissermaßen über den Konkurrenzen und Kalkülen, die dort von Bedeutung sind. Risiken des „Trittbrettfahrens" der Kooperationspartner und der privaten Aneignung öffentlich erstellter Güter sind für die Individuen wenig relevant.

Dieses risikoarme Handlungsfeld stellt das Ergebnis einer Konfiguration aus technischen, sozialen und institutionellen Innovationen dar, in dem eine bestimmte Form kreativen Handelns nicht nur betrieben wird, sondern man sich auch ihrer Voraussetzungen und der Bedingungen ihrer Reproduktion versichert. Diese erfordert bei aller Freiwilligkeit einen gewissen Ressourceneinsatz, sei es als Nebeneffekt und Umwidmung anderer Aktivitäten, durch die Spielräume und Reserven, die Or-

ganisationen für Innovationen sowieso vorhalten müssen, durch staatliche Förderung, privatwirtschaftliches Sponsoring und darüber hinaus durch eine Kultivierung und Reflexion der sozialen, institutionellen und normativen Voraussetzungen.

Ob aber FS/OS vom Modell für Spezialisten oder Virtuosen (Helmers 1998) zum Modell der Demokratisierung von Technik und Kreativität wird, ist m. E. sozial noch nicht entschieden. Es wird auch in Zukunft davon abhängen, wo sich vergleichbare Konfigurationen aus vernetzter Produktion, öffentlichen Gütern und Handelnden entwickeln, die sich ihrer Mittel und Ressourcen bemächtigen und ihre Spielräume und Kompetenzen erweitern. Es wird darauf ankommen, dass in diesen Konfigurationen und gesellschaftsweit der Anspruch auf Gestaltung von Technik und Lebensverhältnissen sich nicht auf die ExpertInnen beschränkt. Um die Ausblendungen und sozialen Exklusivitäten einer reibungslos implementierten Moderne zu überwinden, müsste das heterogene Engineering noch heterogener werden.

Literatur

Beckert, Jens (1997): *Grenzen des Marktes*, Frankfurt am Main/New York.

Beirne, Martin, Harvie Ramsay, Androniki Panteli (1998): *Developments in Computing Work: Control and Contradiction in the Software Labour Process*. In: Thompson, P., C. Warhurst (Hg.): Workplaces of the Future. Houndsmills, S. 142–162.

Benhabib, Seyla (1992): *Situating the Self. Gender, Community, and Postmodernism in Contemporary Ethics*, London/New York.

Benkler, Yochai (2002): *Coase's Penguin, or Linux and the Nature of the Firm,* Yale Law Journal 112,
online http ://www.benkler.org/CoasesPenguin.pdf (9.8. 2002).

Castells, Manuel (1996): *The rise of the network society*, Oxford.

Castells, Manuel (2001): *The Internet Galaxy. Reflections on the Internet, Business, and Society*, Oxford, New York.

De Certeau, Michel (1988): *Kunst des Handelns*, Berlin.

Elster, Jon (1987): *Subversion der Rationalität*, Frankfurt am Main/New York.

Fraser, Nancy (1994): *Widerspenstige Praktiken. Macht, Diskurs, Geschlecht*, Frankfurt am Main.

Ghosh, Rishab A. / Glott, Rüdiger / Krieger, Bernhard / Robles, Gregorio (2002): *Free/Libre and Open Source Software: Survey and Study*, Deliverable D 18: Final Report, Part 4: Survey of Developers,
online http://floss1.infonomics.nl/finalreport/FLOSSfinal-4.pdf
(25. 7. 2002).

Ghosh, Rishab Aiyer / Prakash,Vipul Ved (2000): *The Orbiten Free Software Survey 1st edition*, May 2000,
online http://orbiten.org/ofss/01.htm.

Glißmann, Wilfried (2002): *Der neue Zugriff auf das ganze Indiciduum. Wie kann ich mein Interesse behaupten?* In: Moldaschl, M. / Voss, G. G. (Hg.): Subjektivierung von Arbeit. München, Mering, S. 241–259.

Gorz, André (2002): *Welches Wissen? Welche Gesellschaft?* In: Heinrich-Böll-Stiftung (Hg.): Gut zu wissen. Links zur Wissensgesellschaft. Münster, S. 14–35.

Grassmuck, Volker (2000): *Freie Software. Geschichte, Dynamiken und gesellschaftliche Bezüge*,
online http://mikro.org/Events/OS/text/freie-sw.html (13.8.2003).

Habermas, Jürgen (1971): *Vorbereitende Bemerkungen zu einer Theorie der kommunikativen Kompetenz*. In: Habermas, Jürgen / Luhmann, Niklas (Hg.): Theorie der Gesellschaft oder Sozialtechnologie. Was leistet die Systemforschung? Frankfurt am Main, S. 101–141.

Helmers, Sabine (1998): *Digitale Hahnenkämpfe. Zur Ethnographie der Computer-Virtuosen*. In: Fröhlich, G. / Mörth, I (Hg.): Symbolische Anthropologie der Moderne. Kulturanalysen nach Clifford Geertz. Frankfurt am Main, S. 139–148.

Hertel, Guido / Niedner, Sven / Herrmann, Stefanie (2003): *Motivation of software developers in open source projects: An internet-based survey of contributors to the Linux kernel*, Research Policy 32, S. 1159–1177.

Himanen, Pekka (2001): *The hacker ethic and the spirit of the information age*, New York.

Holtgrewe, Ursula (2001): *Kreativität als Norm – zum Erfolg verdammt? Open-Source-Software zwischen sozialer Bewegung und technischer Innovation*. In: Allmendinger, J. (Hg.): Gute Gesellschaft? Verhandlungen des 30. Kongresses der Deutschen Gesellschaft für Soziologie in Köln 2000. Opladen, S. 399–424.

Holtgrewe, Ursula (2004): *Articulating the Speed(s) of the Internet: The Case of Open Source/Free Software*, erscheint in Time & Society 13.

Holtgrewe, Ursula / Werle, Raymund (2001): *De-commodifying software? Open Source software between business strategy and social movement*, Science Studies 14, S 43–65.

Kalkowski, Peter / Mickler, Otfried (2002): *Zwischen Emergenz und Formalisierung – Zur Projektifizierung von Organisation und Arbeit in der Informationswirtschaft*, SOFI-Mitteilungen 30, S. 119–134.

Lakhani, Karim R. / Wolf, Bob / Bates, Jeff / Dibona, Chris (2002): *The Boston Consulting Group Hacker Survey release 0.73*,
online http://www.osdn.com/bcg/BCGHACKERSURVEY-0.73.PDF (5.8.02).

Lakhani, Karim / von Hippel, Eric (2000): *How Open Source software works: „Free" user-to-user assistance*, MIT Sloan School of Management Working Paper #4117,
online http://opensource.mit.edu/online_papers.php (5.2.2002.)

Landley, Rob / Raymond, Eric S. (2003): *Halloween IX: It Ain't Necessarily SCO: Revision 1.0*,
online http://www.opensource.org/halloween/halloween9.htm.

Lave, Jean / Wenger, Etienne (1991): *Situated learning – Legitimate Peripheral Participation*, Cambridge.

Law, John (1987): *Technology and heterogenous engineering: the case of the Portuguese expansion*. In: W. E. Bijker / Hughes, T. P. / Pinch T. (Hg.): The social construction of technological systems. New directions in the sociology and history of technology. Cambridge, Mass., S. 111–134.

Marx, Karl (1858, 1974): *Grundrisse der Kritik der politischen Ökonomie*, Berlin.

Meretz, Stefan (2000): *LINUX & CO. Freie Software – Ideen für eine andere Gesellschaft. Version 1.01*, letzte Änderung: 03.07.2000, online http://www.kritische-informatik.de/fsrevol.htm.

Moody, Glyn (2002): *Rebel Code. Linux and the open source revolution*, London.

Moon, Jay Yun /Sproull, Lee (2000): *Essence of distributed work: The case of the Linux kernel*, First Monday 5, online http://www.firstmonday.org/issues/issue5_11/moon/index.html (12. 11. 2000).

O'Mahony, Siobhan (2003): *Guarding the commons: How community managed software projects protect their work*, Research Policy 32, S. 1179–1189.

Ortmann, Günther (1995): *Formen der Produktion. Organisation und Rekursivität*, Opladen.

Rammert, Werner (1988): *Das Innovationsdilemma. Technikentwicklung im Unternehmen*, Opladen.

Rammert, Werner (1997): *Innovation im Netz. Neue Zeiten für technische Innovationen: heterogen verteilt und interaktiv vernetzt*, Soziale Welt 48, S. 397–416.

Raymond, Eric (1998): *The cathedral and the bazaar*, First Monday 3, online http://www.firstmonday.org/issues/issue3_3raymond/index.html (25. 7. 2000).

Raymond, Eric (1999a): *Homesteading the Noosphere*, Version August 1999, online http://www.tuxedo.org/~esr/writings/homesteading/homesteading.txt (25. 7. 2000).

Raymond, Eric (1999b): *Shut up and show them the code*, Version 9.7.1999, online http://www.tuxedo.org/~esr/writings/shut-up-and-show-them.html (25. 7. 2000).

Robles, Gregorio / Scheider, Hendrik / Tretkowski, Ingo / Weber, Niels (2001): *Who is Doing It? A research on Libre Software developers*, version 27.8.2001, online http://widi.berlios.de , figures http://widi.berlios.de/stats.php3.

Sauer, Dieter / Lang, Christa (Hg.) (1999): *Paradoxien der Innovation. Perspektiven sozialwissenschaftlicher Innovationsforschung*, Frankfurt am Main/New York.

Smith, Dorothy E. (1987): *The everyday world as problematic*, A feminist sociology.

Tuomi, Ilkka (2002): *Networks of innovation. Change and meaning in the age of the Internet*, Oxford.

Weber, Max (1921, 1980): *Wirtschaft und Gesellschaft*, Tübingen.

Weyer, Johannes / Kirchner, Ulrich / Riedl, Lars / Schmidt, Johannes F. K. Hg. (1997): *Technik, die Gesellschaft schafft. Soziale Innovation als Ort der Technikgenese*, Berlin.

Winzerling, Werner (2002): *Linux und Freie Software. Eine Entmystifizierung*, prokla 126, S. 37–55.

Open Source und Freie Software – soziale Bewegung im virtuellen Raum?

THOMAS ZIMMERMANN

1. Einleitung – Wer bewegt wen oder was?

Seit Beginn der 1980er Jahre hat sich im Bereich der Software, ausgehend von einer Initiative einiger Softwareentwickler, ein Phänomen etabliert, das bis heute eine kurze, aber bewegte Entwicklungsgeschichte aufweisen kann: Freie Software *(Free Software – FS)* und seit 1998 Offene Quellen *(Open Source – OS)*. Die Vordenker der an diesem Prozess beteiligten Programmierer nehmen dabei nicht weniger für sich in Anspruch, als eine soziale Bewegung zur Erhaltung und Durchsetzung von essenziellen Grundfreiheiten im virtuellen Raum zu sein. Diese Freiheiten definiert der Gründer der *Free Software Foundation* (FSF) Richard Stallmann wie folgt:

1. Die Freiheit zur uneingeschränkten Nutzung von Software
2. Die Freiheit zur Veränderung und Anpassung von Software durch offene Quellen
3. Die Freiheit zur Verteilung von kostenpflichtigen oder kostenlosen Kopien von Software
4. Die Freiheit zur Verteilung veränderter Versionen (vgl. Stallman 1999).

Im Angesicht der historischen Entwicklung sozialer Bewegungen zu einer gesellschaftspolitischen Formation mit bis heute bedeutender Funktion für „politischen Wandel von Unten" in Demokratien soll dieser Anspruch hier einer sozialwissenschaftlichen Überprüfung unterzogen werden.

Für den Themenkomplex „soziale Bewegung" sind die Sozialwissenschaften zuständig. Diese existieren bereits länger als die Informatik. Im Vergleich zu den Naturwissenschaften sind sie jedoch ebenfalls eine verhältnismäßig junge Disziplin. So erklärt sich auch, dass die Bewegungsforschung als Fachrichtung für die Untersuchung sozialer Bewegungen erst zu Beginn des 20. Jahrhunderts entstand. Sie ist ursprünglich als der Versuch zu interpretieren, das für den damaligen Zeitraum politisch bedeutendste Phänomen der sozioökonomischen Entwicklung zu erklären: die Arbeiterbewegung – „die Mutter der sozialen Bewegung".

Die Industrielle Revolution hatte zu diesem Zeitpunkt über einen Zeitraum von gut 100 Jahren die wirtschaftlichen und gesellschaftlichen Strukturen Europas grundlegend verändert. Die zentrale technische Innovation dieser Zeit war die mechanische Maschine, die, angetrieben durch die Verbrennung fossiler Energieträger, den Kern einer neuen ökonomischen Ära stellte. Diese Entwicklung hatte mit zeitlicher Verzögerung einschneidende Auswirkungen auf die ehemals durch agrarische Verhältnisse bestimmte Lebenswelt eines Großteils der Bevölkerung Europas. Urbanisierung und Proletarisierung weiter Bevölkerungsschichten sind in diesem Zusammenhang als direkte Folgen des hohen Bedarfs an Maschinen-Arbeitern zu ver-

stehen, der durch rasch wachsende Industrien unterschiedlichster Art entstand. Der produktionsbedingte Umstand der räumlichen Konzentration zu Fabrikstandorten sollte den Industrie-Arbeitern jedoch gegenüber den verelendeten Menschen in agrarischer Lebensweise (Bauern, Tagelöhner, Fronarbeiter etc.) einen entscheidenden Vorteil bescheren: Sie konnten sich mit dem Ziel der Verbesserung der eigenen Lebensverhältnisse besser organisieren.

Damit sind schon die ersten notwendigen Kriterien zur Feststellung einer sozialen Bewegung benannt und es wird eine grundlegende Definition sozialer Bewegungen fällig. Diese Definition soll im Verlauf dieses Artikels zur Beantwortung der Ausgangsfrage herangezogen werden: „Soziale Bewegung ist ein mobilisierender kollektiver Akteur, der mit einer gewissen Kontinuität auf der Grundlage hoher symbolischer Integration und geringer Rollenspezifikation mittels variabler Organisations- und Aktionsformen das Ziel verfolgt, grundlegenderen sozialen Wandel herbeizuführen, zu verhindern oder rückgängig zu machen“ (Raschke 1988, S. 77). Diese allgemeine Definition beschreibt den Status der Arbeiterbewegung bis Ende des 19.Jh. in Deutschland nahezu perfekt.

Was hat das mit Freier Software und Open Source zu tun, fragt sich der ein oder andere Leser vielleicht? Nun, die Arbeiterbewegung soll im Folgenden als historisches Beispiel zur Veranschaulichung der sozialwissenschaftlichen Theorieelemente der Bewegungsforschung dienen. So soll dem Leser ein konkreter Maßstab zur Beurteilung der These, FS/OS wäre eine soziale Bewegung der Informationsgesellschaft, an die Hand gegeben werden. Dabei kann natürlich die Zielsetzung prominenter Akteure aus dem FS/OS-Bereich nicht als Bewertungskriterium verwandt werden. Vielmehr müssen Strukturen und Aktivitäten von FS/OS untersucht und ihr Verhältnis zu den anderen relevanten Teilen der Gesellschaft hinterfragt werden. „Es erscheint deshalb angemessener, Bewegungen auf inhaltlicher Ebene nicht durch einen bestimmten politischen Zielkatalog, sondern ihren allgemeineren Bezug auf Gesellschaft zu definieren. Soziale Bewegungen sind *soziale* Bewegungen, weil sie Gesellschaft für gestaltbar halten und auf deren Grundstrukturen einzuwirken suchen“ (Rucht 1997, S. 391). Mittels der Ergebnisse der Bewegungsforschung und einer dementsprechend ausgerichteten Betrachtung von FS/OS wird im Folgenden die Ausgangsfrage einer theoretischen Überprüfung unterzogen.

2. Kollektive, gerichtete Aktion – Prüfung der Basiselemente sozialer Bewegungen

Seit dem Entstehen der Bewegungsforschung haben sich Forscher auf Micro-, Meso- und Makroebene um die Identifikation und Beschreibung der zentralen Elemente sozialer Bewegungen bemüht. Eine endgültige Sicht der Dinge existiert hier (wie in fast allen Bereichen der Gesellschaftswissenschaften) nicht. Trotzdem lässt sich ein weitgehend etablierter Katalog an Strukturelementen und Bedingungen zusammenstellen, der es ermöglichen sollte, die Ausgangsfrage im Rahmen dieses Artikels befriedigend zu beantworten.

Beginnen wir mit der oben bereits genannten allgemeinen Definition. Die Arbeiterbewegung des 19. Jahrhunderts erfüllte diese Definition zu Beginn in einem Rah-

men, der von unserem modernen Bild zu trennen ist. Damals stellten die Arbeiter den größten Teil der städtischen Bevölkerung Europas. Eine politische Interessenvertretung existierte jedoch noch nicht. Deshalb zeigte die Arbeiterbewegung zu Beginn vor allem Formen der Selbsthilfe: Gewerkvereine, Gewerksgenossenschaften und Arbeiterschaften gehörten zu den ersten Organisationsformen, die mit den Methoden der Solidarkasse (ähnlich den heutigen Kranken- und Sozialkassen) und des Bildungsvereins (Bildung war für Arbeiter und deren Kinder kulturelles Neuland) etc. umgesetzt wurden. Die bis heute wirksame Idee des Sozialstaates basiert auf diesen frühen Formen der Solidargemeinschaft (vgl. Lampert 1998, S. 64ff). Die strukturelle Benachteiligung der Arbeiterschaft in der Gesellschaft gegenüber den neuen, bürgerlichen Mächtigen sollte durch kollektive Aktion und Solidarität bekämpft werden. Eine direkte oder vermittelte Einflussmöglichkeit auf das politische Geschehen war zu diesem Zeitpunkt auf Grund der früh- bzw. vordemokratischen Staatsformen (in der Regel konstitutionelle Monarchien) nicht möglich, wohl jedoch die gemeinsame Aktion auf ökonomischer Ebene, die trotz der fehlenden politischen Ausrichtung durchaus als Protestform zu werten ist. Die Quasi-Monopolisierung der Lohnaushandlung durch gewerkschaftliche Vertretungen erleichterte die Einkommenssituation, und die auf die private Lebenswelt gerichteten Initiativen verbesserten die kulturellen und sozialen Bedingungen des Alltags. Die Kontinuität dieser Aktionen drückt sich vor allem in den auf längere Zeitspannen ausgelegten Organisationsformen aus. Eine Sozialkasse beispielsweise ist keine ad-hoc-Form sozialer Aktivität wie eine Demonstration. Hier ist eine regelmäßige und längerfristige Teilnahme der Mitglieder die Voraussetzung dafür, dass sich der soziale Nutzen in Form der gesteigerten sozialen Absicherung gegen unvorhergesehene Schicksalsschläge wie Krankheit, Invalidität etc. erzielen lässt. Damit ist auch ein gewisser Grad an organisatorischen Strukturen notwendig, um diese Aktivitäten der sozialen Bewegung zu koordinieren.

Die Mobilisierung der Mitglieder erfolgte durch die Konstruktion von Gemeinsamkeit mittels Sprache, Kleidung, Musik und Abzeichen als Formen symbolischer Integration. In Verbindung mit der kontinuierlichen Thematisierung der kollektiven Benachteiligung wurde so eine kollektive Identität – ein ausgeprägtes „Wir-Gefühl" zur Abgrenzung vom „Gegner" – geschaffen und gepflegt. Im Rahmen dieser kollektiven Identität lag die spezielle Wahl der Teilnahmeform weitgehend in der Hand der Mitglieder. Der mittlere Organisationsgrad und die damit relativ flache Struktur der Arbeitsteilung in der Bewegung ließ den Anhängern den Freiraum, sich ihr Betätigungsfeld im Rahmen der gemeinschaftlichen Aktionen weitgehend selbst zu gestalten. Wollte sich eine kleine Gruppe von Akteuren z.B. spontan mit den Arbeitsbedingungen in einem bestimmten Betrieb befassen, konnte sie dies tun, ohne mit Problemen durch konkurrierende Akteure in den eigenen Reihen kämpfen zu müssen. Trotz eines Mindestgrades an kollektiver Organisation waren die Einzelteile der Bewegung untereinander nicht streng hierarchisch angeordnet. Diese organisatorische Flexibilität ist gemeinsam mit der geringen Rollenfestlegung der einzelnen Akteure (jeder kann Kassenwart, aber auch Straßenkämpfer sein) eine wichtige Voraussetzung, die Menschen für gemeinsame Aktionen im Rahmen sozialer Bewegungen nachhaltig zu mobilisieren (zur Theorie und Geschichte der Arbeiterbewegung vgl.

Müller-Jentsch 1985, S. 50 ff.; Kocka 2001, S. 8 ff.). Bis hier sollte die allgemeine Definition einer sozialen Bewegung mit Realitätsbezug gefüllt worden sein.

2.1. Ist FS/OS eine soziale Bewegung?

Nun soll eine erste Übertragung auf den FS/OS-Bereich versucht werden. Finden sich die bei Raschke (s.o.) definierten notwendigen Bedingungen einer sozialen Bewegung von mobilisierender, kollektiver Aktion, Kontinuität, kollektiver Identität, mittlerem Organisationsgrad, geringer Rollenspezifikation und protestförmigem Willen zum sozialen Wandel auch im Rahmen des FS/OS-Phänomens wieder? Am leichtesten fällt diese Zuordnung für die kollektive, protestförmige Aktion. Die an anderer Stelle des vorliegenden Jahrbuches detailliert dargestellte Entstehung und Entwicklung von Freier Software seit den 1980er und von Open Source seit den 1990er Jahren ist vor allem durch zwei Aspekte gekennzeichnet:

Zu nennen wären zunächst der Protest und der Widerstand gegen die fortschreitende Privatisierung[1] bereits geleisteter und zukünftiger Innovationen im Software-Bereich durch das neuartige Instrument der freien Lizenzen sowie die organisierte, gemeinschaftliche Entwicklung wichtiger Softwarekomponenten zur Herstellung eines freien Betriebssystems und der dazugehörigen Anwendungen, um proprietäre Software so weit als möglich zu ersetzen. Dies gilt vor allem für die Mobilisierungsakteure der FSF.

In Zusammenhang damit steht ein zweiter Aspekt: Die Entstehung einer wirklich globalen Entwickler-Anwender-Gemeinde aus den elitären „Hacker-Strukturen“[2] der Frühphase der wissenschaftlichen Informatik heraus. Verstärkt durch das explosionsartig wachsende Internet hat die Hackerkultur eine eigene Sprache, neue Kommunikationsformen, virtuelle Treffpunkte, Normen und Regeln entwickelt. So ist eine hoch mobilisierende Organisationsstruktur mit schwacher bis mittlerer Regelungsdichte zur Forcierung der gemeinsamen Ziele entstanden.[3]

Die mobilisierende, kollektive Aktion auf der Basis eines geteilten Identitätsmusters bei mittlerem Organisationsgrad lässt sich für den FS/OS-Bereich also durchaus feststellen (vgl. Stallman 1992; Raymond 2001, S. 1 ff, S. 167 ff; Grassmuck 2002; Greve 2003).

Darüber hinaus ist der zentrale Punkt der gemeinsamen politischen Ziele der Bewegung zu hinterfragen. Zwar wird die Sicherung sozialer Gerechtigkeit im Kontext des „Informationszeitalters“ von der FSF als eines der Hauptziele freier Software deklariert. Die entscheidende Frage besteht jedoch darin, ob die Strukturen der entstehenden Informations- bzw. Wissensgesellschaft dieser Wahrnehmung

---

1 Im Sinne des englischen Begriffs „proprietery software“ als Bezeichnung für kommerzielle, über Lizenzeinnahmen finanzierte und durch Eigentumsrechte geschützte Software wird im Weiteren der Begriff „proprietäre Software“ verwendet.

2 Der Hacker-Begriff ist nicht in seiner populären Bedeutung eines Computerkriminellen gemeint, sondern in seiner ursprünglichen Bedeutung zur Bezeichnung eines talentierten und vor allem passionierten Programmierers (vgl. Levy 1984).

3 Die weitläufigen und sehr interessanten Hintergründe zum Begriff des Allgemeingutes bzw. der Allmende können hier aus Platzgründen nicht erfolgen. Die Möglichkeiten zu Kommunikation, Koordination und kooperativer Produktion werden in anderen Artikeln dieses Bandes ausführlich besprochen (vgl. Ostrom 1991).

auch entsprechen? Mit anderen Worten: Ist die Art und Form der Herstellung, Verteilung und Nutzung von Software eine Frage von gesellschaftlicher Bedeutung? Und zielt FS/OS damit auf einen sozialen Wandel ab? Eine Antwort auf diese zentrale Frage kann im Rahmen dieses Artikels nur zusammenfassend und damit verkürzt geleistet werden. Zunächst ist dazu die Rolle von Software in einem sozioökonomischen Modell der Gegenwart zu klären, um im Anschluss daran die neue bzw. strukturverändernde Bedeutung von FS/OS für Software im Allgemeinen zu diskutieren.[4]

2.2. Software in der sozioökonomischen Struktur der Gegenwart

Grundlegend ist trotz geplatzter New Economy-Blase und weltwirtschaftlicher Rezession durchaus von einer neuen, informations- bzw. wissenszentrierten Form der Wertschöpfung in Teilen der Wohlstandsökonomien auszugehen.[5] Wissen ist dabei als individuell gültiges Konzept kognitiver Strukturen zu verstehen, das durch den Akt der Kodifizierung (Umsetzung in eine beliebige Kommunikationsform) zu allgemein verwendbaren Informationen transformiert wird.

Der Trend zur Wissenszentrierung der Wirtschaft lässt sich einerseits rein quantitativ begründen: Die Bedeutung produktbezogener Arbeit geht in den OECD-Ländern langfristig zurück und es besteht ein eindeutiger Trend zur Steigerung der technologischen Intensität des Weltgüterhandels. Die Volumina von informationsbezogener Arbeit, Produktion und Handel steigen also weltweit an.

Andererseits liegen auch gute qualitative Gründe vor, eine neue Entwicklungsphase im ökonomischen Prozess in diesem Zusammenhang anzunehmen. Die Informations- und Kommunikationstechnologien (IuKT) sind als Querschnittstechnologien zu betrachten, deren zunehmende Nutzung in allen denkbaren Bereichen der Wirtschaft eine bedeutende Veränderung gegenüber dem spät-industriellen, fordistischen Entwicklungsmodus darstellt. Diese Bedeutung entspringt dem Charakter der universellen Nutzbarkeit dieser Technologien.[6]

Damit bilden die IuKT den Kern eines neuartigen soziotechnologischen Paradigmas[7], in dessen Rahmen sich digitalisiertes Wissen neben menschlicher Arbeit und natürlichen Ressourcen zu einem weiteren Strukturelement der Kapitalakkumu-

[4] Die folgenden Feststellungen stellen die Ergebnisse einer ausführlichen Untersuchung der neuen Qualität gesellschaftlicher Entwicklung unter den Auswirkungen von Informations- und Kommunikationstechnologien dar. Die Begründung der einzelnen Aussagen zu Entwicklungsmodus, dem zugehörigen sozio-technologischen Paradigma und dem damit verknüpften Wachstumsregime kann hier aus Platzgründen nicht erfolgen (vgl. Zimmermann 2002).

[5] Dies wird von Beobachtern aus dem neomarxistischen Lager gerade andersherum wahrgenommen. Dort wird FS/OS als Teil des New-Economy-Börsenhypes verstanden. Einige Autoren folgern daraus das Scheitern von FS/OS: „Die ideologisch motivierten Versuche, eine solche eigentumslose kapitalistische Entwicklung zu begründen (maßgeblich: Raymond 1999), sind in der Praxis grandios gescheitert – sei es im Bereich des freien Betriebssystems Gnu/Linux oder in der Musikindustrie“ (Gröndahl 2002, S. 100). Gegen diese Wahrnehmung spricht die Realität. FS/OS Projekte wachsen und gedeihen nach wie vor.

[6] Die ähnelt der universellen Nutzbarkeit der mechanischen Maschine, die als Querschnittstechnologie des industriellen Entwicklungsmodus zu verstehen ist.

[7] Ein soziotechnologisches Paradigma bezeichnet eine spezifische Konstellation von gesellschaftlichen Organisationsformen und technologischen Strukturen (vgl. Hübner 1989, S. 140ff; Castells 2000, Grahl/Teague 2000, S. 161).

lation entwickelt (vgl. Castells 2000, S. 17ff). Arbeit und Ressourcen werden im industriellen Modus mittels Wissen derartig kombiniert, dass bei geringstem Mitteleinsatz innerhalb eines zu optimierenden Produktionsprozesses ein maximaler Wert des Endproduktes realisiert werden kann. Wissen hat also auch hier eine wichtige Bedeutung im Rahmen der Gestaltung und Optimierung ökonomischer Abläufe.

Der entscheidende Unterschied zur Vergangenheit liegt in der selbstreferenziellen Bedeutung, die Wissen in zunehmendem Maße hat. In unseren Wohlstandsökonomien können wir nach 150 bis 200 Jahren industrieller Entwicklung nahezu alle physischen Produkte in beliebiger Skala zu geringen Kosten ohne nennenswerten Einsatz von menschlicher Arbeit herstellen. Damit erlangen Produkte und Dienstleistungen „virtueller Natur“[8] eine immer stärkere Bedeutung, da diese einerseits einen erweiterbaren Raum für menschliche Beschäftigung bieten und andererseits eine weitere Optimierung der physisch ausgereizten Produktionsverfahren ermöglichen.

Darüber hinaus kann digitalisiertes Wissen mit Hilfe der IuKT je nach Einsatzbereich alle wichtigen Stationen eines Wirtschaftsprozesses bestimmen: Es kann alternativ Ressource, Prozess und Produkt oder auch alles auf einmal sein. Für quasi rein informationsbezogene Wirtschaftsbereiche, wie beispielsweise Softwarebranche sind damit einige grundlegende Annahmen der physischen Ökonomie (z.B. absolute Knappheit, konkurrierende Nutzung, Transaktionskosten) in Frage zu stellen. Im informationalen Verwertungsprozess verlieren Arbeit und natürliche Ressourcen zwar nicht ihre Bedeutung, ihr Stellenwert ändert sich jedoch durch die Einbeziehung des neuen Produktionsfaktors digitalisiertes Wissen. Dessen Nutzung im ökonomischen Streben nach der Realisierung des effizientesten Verwertungsprozesses schafft andere Implikationen für wirtschaftliches Verhalten, als dies im physischen Raum der Fall ist (vgl. McFadden 1998; Arthur 1994, 1999a+b, 2000; Castells 2000, S. 71 ff., Christensen 2000, S. 257 ff.; Lessig 2001, S. 26 ff.).

Produktivität und Wettbewerbsfähigkeit der ökonomischen Akteure hängen bedeutend von deren Fähigkeit zur effizienten Erzeugung, Akkumulation, Behandlung, Anwendung und Reproduktion von digitalisiertem Wissen in Verbindung mit anderen Produktionsfaktoren ab. Wissen und Information werden selbst zum Gegenstand ökonomischer Verwertung und unterliegen dabei anderen, virtuellen Bedingungen. Diese neuartige Konstellation von Wirtschaftsfaktoren wird im Weiteren als *informationaler Entwicklungsmodus* bezeichnet und lehnt sich damit in der Terminologie an die sozioökonomische Theorie der Regulationsschule an (vgl. Krieg 1998; Zimmermann 2002, S. 14ff; Hübner 1989). Mit der digitalen Form der Information und der breiten Diffusion der notwendigen Technologien verändern sich so einerseits die bisherigen Faktoren der Kommunikation, der physischen Produktion und der Wissenserzeugung (sei es öffentliche oder private Forschung). Andererseits entstehen neue soziale und ökonomische Formen der Interaktion, deren Ablauf und Gegenstand nahezu vollständig in der digitalen Form besteht (virtuelle Interessengemeinschaften, Marktplätze, Produkte und Unternehmen) und damit weitgehend von physischen Begrenzungen frei ist.

[8] „Virtuell“ ist hier als Negation von „physisch“ zu verstehen.

Ein wichtiges strukturelles Element stellt in diesem Zusammenhang die Software dar. Eine Software realisiert einen definierten Algorithmus zur automatischen, halbautomatischen und interaktiven Erzeugung, Bearbeitung und Wiedergabe von Daten (also in strukturierter Form auf einem IuKT-Gerät vorliegende Informationen) aller Art. Sie ist damit als digitalisiertes Prozess- bzw. Objektwissen zu verstehen.[9] Software kann entweder in Form des für den Menschen verständlichen Quellcodes in einer Programmiersprache oder in einer für den Computer „verständlichen" und damit als Abfolge von Befehlen ausführbaren Form in Maschinensprache vorliegen. Damit stellt sie die zentrale Schnittstelle zwischen Maschine, Daten und Benutzer dar. Jeder Mensch, der mit digitalisierten Daten etwas anfangen möchte, ist auf die Benutzung von Software angewiesen. Dies bedeutet, dass Software direkt den menschlichen Handlungsspielraum bei der Abfrage, Speicherung und Nutzung digitaler Daten bestimmt. Das betrifft nicht nur die technischen Funktionen, die eine Software realisiert, sondern auch die Art der Daten, die durch die Software genutzt werden sollen bzw. können. Der Hersteller einer Software ist also in der Lage, die Datenzugriffe der Benutzer durch die Funktionen seiner Software zu strukturieren und zu steuern.

2.3. Die sozialen Dimensionen von FS/OS

Dieser Schnittstellencharakter von Software und die Annahme eines sich gegenwärtig entwickelnden, informationalen Entwicklungsmodus stellen die entscheidenden Momente bei der Beantwortung der Frage nach grundlegenden Interdependenzen zwischen den gesellschaftlichen und den technologischen Aspekten dar. Werden beide Momente grundsätzlich als gegeben akzeptiert, lässt sich in der Folgerung das Argument vertreten, dass die sozioökonomische Struktur der Herstellung, Verteilung und Nutzung von Software einen bedeutenden Einfluss auf die Lebensqualität und Lebenschancen der Mitglieder unseres gegenwärtigen Gesellschaftssystems haben. Dieser Einfluss von Computeranwendungen auf den gesellschaftlichen Alltag wird sich mit dem Prozess der weiteren Ausgestaltung des informationalen Entwicklungsmodus weiter erhöhen.

Da Software jedoch nicht nur Dienstleistungs-, sondern ebenso Ressourcen- und Produktcharakter haben kann, wirkt sie in unterschiedlichsten Bereichen des gesellschaftlichen Systems. Software und die damit verbundenen sozioökonomischen Strukturen beeinflussen so z.B. die Chancen zur Teilnahme in den Subsystemen Bildung, Forschung, Kommunikation und Wirtschaft teilweise direkt, teilweise auch nur vermittelt. Zusätzlich spielt Software im Rahmen der informationalen Entwicklung auch eine Pionierrolle: Die Branche ist die erste, wirklich informationale Wirtschaftsform, da sowohl Inputs als auch Outputs der Softwareentwicklung virtuell sind. Nur die Struktur der bearbeiteten Daten sowie die Ausführungsbedingungen (Speicherplatz, Performance etc.) stehen noch in einer direkten Verbindung zur

[9] Die Form und damit die Qualität des in Software digitalisierten Wissens hängt mit den Möglichkeiten der jeweils genutzten Programmiersprache zusammen. Die gängigste Form ist momentan die objektorientierte Programmierung. Auch die Strukturen digitaler Daten (Dateiformate, Datenbankstrukturen etc.) befinden sich in einer ständigen Entwicklung. Bis heute hat sich eine Abfolge von softwaretechnologischen Paradigmen etabliert, deren Ende nicht abzusehen ist und hier nicht weiter ausgeführt werden kann.

physischen Welt. Diese Branche ist damit als Vorreiter einer wissenszentrierten Wirtschaftsweise zu verstehen. Es ist zu erwarten, dass zukünftige Segmente der informationalen Wirtschaft (z.B. die angewandte Biotechnologie) sich nach ähnlichen Mustern entfalten werden.

Nach diesen Überlegungen sind wir nun für die Beantwortung der Frage nach der gesellschaftlichen Relevanz der FS/OS-Bewegung gewappnet. Wenn Software den eben beschriebenen Stellenwert hat, dann ist damit ein erster wichtiger Schritt zur Beurteilung der Ausgangsthese getan. Denn das FS/OS-Phänomen ist hier entstanden und hat seine Basis von Anhängern und Akteuren seit seinem Auftauchen in den 1980er Jahren deutlich ausbauen können. FS/OS-Software ist seitdem dabei, der proprietären Softwareentwicklung in einigen Marktsegmenten das Wasser abzugraben. Proprietäre, privatisierte Formen von Kontrolle über Software sowie deren Entstehung und Nutzung sollen durch FS/OS verdrängt werden.

An dieser Stelle überschneiden sich die Ziele von FS/OS mit jenen der kollektiven Identität. Hier entwickelt sich das Bild einer monopolisierten IT-Industrie als Gegner von FS/OS, dessen Bedeutung von Sprechern der Bewegung zwar gern heruntergespielt wird, welches jedoch gerade im Prozess der Identitätsstiftung einer sozialen Bewegung von hoher Bedeutung ist. Und dieser Gegner heißt heute Microsoft[10], personifiziert durch den beliebtesten Buh-Mann der Softwarewelt: Bill Gates. Die Identitätsmuster von FS/OS gehen jedoch über eine schlichte Anti-Haltung deutlich hinaus. Aus den Anfängen der Hackerkultur entwickelten sich diverse spezifische Sprachformen (vgl. Jargon File 4.0.0), Normen und Regeln, die den Teilnehmern eine pragmatische und attraktive Identität mit hohem Mobilisierungspotential bietet. „The hacker culture also, arguably, prefigures some profound changes in the way humans will relate to and reshape their economic surroundings. This should make what we know about the hacker culture of interest to anyone else who will have to live and work in the future" (Raymond 2001, S. XII).

Die kollektive Identität als Bedingung für eine soziale Bewegung ist demnach ebenfalls in hinreichendem Maße gegeben. Damit ist aus der theoretischen Basisdefinition der kollektiven, gerichteten Aktion nur noch ein Element zu betrachten: Der Protest. Jenseits des kommunizierten Protestes im Rahmen der modernen Mediengesellschaft basiert OS/FS auf einer recht ungewöhnlichen Protestform, die direkt mit der „gegnerischen" Sozialstruktur, der proprietären Software, verbunden ist: der Lizenz. Trotz gravierender Unterschiede zwischen den gängigen FS/OS-Lizenzmodellen in der Praxis ist diesen Lizenzen eines gemeinsam: Sie enthalten antiexklusive Elemente, die gewährleisten, dass die unter ihnen veröffentlichte Software lizenzkostenfrei und in ihrem gesamten Quellcode weitergeben werden muss. Damit ist das digitalisierte Wissen aller Autoren von FS/OS-Software stets für andere Entwickler und damit die Allgemeinheit einsehbar, nachvollziehbar und nutzbar.

Auf diese Weise soll eine vollständige Privatisierung des Wissens verhindert werden. Der Grund liegt auf der Hand: In der Praxis führt die private Kontrolle über

[10] Das war nicht immer so. In den 1980er Jahren hatte diese Position nämlich die heute OS freundliche IBM inne. Da war Microsoft nur ein Wettbewerber von vielen auf dem Softwaremarkt und die IBM hatte durch ihren „First-Mover-Advantage" bei der Systemarchitektur PC die Rolle des bösen Monopolisten inne.

das Wissen bzw. die Information häufig zu einer künstlichen Verknappung und damit zu hohen Preisen sowie zu eingeschränkter Zugänglichkeit. Als Mittel hierzu dient den „Besitzern" bzw. Anbietern die Lizenz, die fast beliebige Beschränkungen der Nutzung von Informationsgütern zulässt. Die FS/OS-Bewegung versucht nun, die seit Beginn der 1980er Jahre zunehmende Privatisierung des zentralen, neuen Produktionsfaktors im informationalen Entwicklungsmodus und die sich daraus ergebenden negativen Folgen für die engagierten Akteure und die Allgemeinheit zu revidieren (vgl. Weber 2000; Lessig 2001, S. 49ff; Grassmuck 2002). Sie bedient sich dabei der gleichen Mittel, nämlich wiederum der Lizenz, die jetzt in ihrer offenen Form aber das Gegenteil erreichen soll: umfassende Zugänglichkeit zum lizenzierten Wissen und nicht zuletzt seine allgemeine Kontrollierbarkeit.

Die negativen Auswirkungen, gegen die eine als soziale Bewegung verstandene FS/OS-Gemeinde protestförmige Aktionen durchführt, sollen nun noch etwas näher betrachtet werden. Interessanterweise zeigen sich die Nachteile der proprietären Softwarestrukturen für die FS/OS-Akteure zunächst nicht in monetärer Form, wie z.B. im Falle der Arbeiterklasse des 19. und frühen 20. Jahrhunderts. Im Gegenteil: Die etablierten Strukturen der Softwarebranche haben zu einem ausgeprägten Expertenstatus von Softwareentwicklern bei hohen Gehältern geführt.[11] Gegen diese Verdienststrukturen ist der Protest nicht gerichtet. Der Kern ist vielmehr in der Öffnung und Weitergabe der Quellcodes zu sehen und hat damit informationalen Charakter.

Ein Entwickler ist im Rahmen seiner alltäglichen Arbeit häufig damit beschäftigt, sich mit Problemen, Abläufen und Algorithmen auseinander zu setzen, die bereits in ähnlicher Form von anderen entwickelt wurden. Softwareerstellung ist wie jede andere wissenszentrierte Arbeit zu einem großen Teil Rekombinationsleistung in Verbindung mit Erfahrung und Innovation. Mit der Veröffentlichung von Quellcodes unter einer freien oder offenen Lizenz (wie z.B. der GPL) entsteht in Verbindung mit dem universellen Kommunikationsmedium Internet ein in Raum, Zeit und Zugang uneingeschränkter Pool von Quellcodes. Diese Codes decken unzählige Arten von Problemen und Inhalten ab und können von anderen Entwicklern angesehen und benutzt werden. Der globale Echtzeitcharakter dieses Pools und die sich daraus ableitenden, kollaborativen Arbeitsmethoden stellen zwei konkrete Neuerungen von Arbeitsmethoden im informationalen Entwicklungsmodus dar. Offene Codes sind demzufolge zum einen eine optimale Möglichkeit zur Erforschung und Erschließung neuer Wissensgebiete durch ausführbare und veränderbare Beispiele, zum anderen bedeuten sie für den allgemeinen Herstellungsprozess von Software eine deutliche Produktivitätssteigerung, da die Wahl der Arbeitsmittel nicht durch eigentumsrechtliche Exklusionsmechanismen eingeschränkt wird. Diese Elemente sind in ihrer Wirkungskraft und ihrer Bedeutung für die Akteure von FS/OS nicht zu unterschätzen, denn ihre Realisierung steigert in erster Linie die Motivation der Entwickler.

Nachhaltige Motivation ist in Bereichen wissenszentrierter Arbeit, die komplexe Zusammenhänge zum Inhalt hat, schwer durch einfache Anreizsysteme wie Leis-

[11] Gute Softwareentwickler bzw. -berater sind nach wie vor teuer, da selten. Die Entwickler aus der OS-Gemeinde bilden in dieser Hinsicht keine Ausnahme.

tungsprämien oder Abgabetermine zu erreichen. Vielmehr lässt sich die Arbeitsmotivation und damit die Effizienz in diesen Bereichen über eine weitestmögliche Gestaltungsfreiheit im Arbeitsprozess erreichen, da die eigenen Entscheidungen zu Erfolg oder Misserfolg führen und somit hauptsächlich die eigenen „Skills" zählen. „Indeed, it seems the prescription for highest software productivity is almost a Zen paradox; if you want the most effecient production, you must give up trying to *make* programmers produce. Handle their subsistence, give them their heads, and forget about deadlines" (Raymond 2001, S. 109).

Zusätzlich zielt FS/OS jedoch auch auf eine mehr allgemeine Ebene als die der Arbeitswelt von IT-Spezialisten. So kommen die Effekte der Softwareentwicklung unter dem FS/OS–Paradigma auf unzähligen Wegen den direkt und indirekt mit Software verbundenen ökonomischen Bereichen zugute. Dieser Schluss ergibt sich aus mehreren Überlegungen. Schätzungsweise 70 bis 90% aller softwarebezogenenen Arbeiten finden innerhalb von Betrieben statt. Das bedeutet, dass hier kein konsumierbares Produkt zum Verkauf hergestellt wird. Viele der erstellten Programme dienen zum Einsatz innerhalb eines Unternehmens, einer Organisation oder eines Netzwerkes. Der überwiegende Teil dieser „In-House" Anwendungen hat dabei keinen wettbewerbskritischen Charakter. Der eigene, geheime Wettbewerbsvorteil spricht also i.d.R. nicht gegen die Nutzung des FS/OS-Paradigmas. Die selbstverstärkende Nutzung von FS/OS in diesen Bereichen lässt einen starken Spill-Over-Effekt auf die Kostenstrukturen des jeweiligen Einsatzgebietes erwarten. Damit könnte sich der Ertrag dieser speziellen In-House-Investitionen auf der Unternehmensebene und damit auch von generellen IT-Investitionen auf der volkswirtschaftlichen Ebene mit der Zeit deutlich verbessern.[12]

Die ersten nachhaltigen Erfolge von Unternehmen mit strategischer Ausrichtung auf FS/OS scheinen für diese Annahme zu sprechen. „Dieses Geschäftsmodell scheint tragfähig zu sein. Tatsächlich meldete IBM kürzlich den Zugewinn zehn weiterer bedeutender Kunden für Linux-basierte Serversysteme. Sie ergänzen die 4600 bestehenden Kunden von IBM, die Linux als Betriebssystem für ihre Web- oder Datenbankserver einsetzen. Um diesen Trend zu verstärken, gewährt IBM seinen Kunden bei der Wahl von Linux als Betriebssystem sogar einen Discount gegenüber anderen Systemen aus dem eigenen Angebot" (Hofmann 2002, S. 6). Darüber hinaus würde ein breiter In-House-Einsatz von FS/OS die gewerblichen Benutzer wieder in die Lage versetzten, ihre Software auszuwählen. Die häufig monopolisierten Strukturen auf den einzelnen Softwareteilmärkten zwingen Unternehmen in ein Technologie-„Lock-In", das ihre strategische Handlungsfähigkeit dramatisch einschränken kann (vgl. Arthur 1994, 1999a+b, 2000). Der Monopol-Hersteller bestimmt dann die Handlungsmöglichkeiten seiner Kunden und nicht umgekehrt. Damit sind nur einige Überlegungen genannt, die in Bezug auf den Einsatz von FS/OS

[12] Bis heute sind die volkswirtschaftlichen Effekte durch die Nutzung von IuKT umstritten. Stichwort hierzu ist das sog. Produktivitätsparadoxon. Dessen Verfechter gehen davon aus, dass der Einsatz neuer Technologien keinerlei produktivitätssteigernde Effekte mit sich bringt (vgl. Gordon 2000, S. 60; Altvater/Mahnkopf 2000, S. 774). FS/OS könnte einen Teil der Lösung dieses Paradoxons darstellen.

auf der Unternehmensebene von Bedeutung sind (für eine weitere Diskussion vgl. Beiträge in diesem Band).

Aber auch Endanwender, die mit Softwareentwicklung nichts zu tun haben, kommen in den Genuss direkter Vorteile von FS/OS, die vor allem im direkt-sozialstrukturellen Kontext in einem informationalen Entwicklungsmodus von Bedeutung sind: Die Chancen zur Teilhabe an gesellschaftlichen Prozessen werden in zunehmendem Maße nicht mehr nur durch den Zugang zu klassischer Bildung vermittelt, wie es in der fordistischen Gesellschaftsstruktur noch der Fall war. Vielmehr ist im informationalen Modus der Zugang zu wertvollen Informationen mit Hilfe von ausgefeilten Softwaretechnologien im Rahmen eines lebenslangen Lernprozesses von entscheidender Bedeutung. Nur durch diesen Modus kann das Individuum sich in unserer zunehmend komplizierter werdenden Welt kontinuierlich das gerade benötigte Wissen aneignen. Und Software ist die Technologie, die Zugang ermöglicht, steuert und begrenzt.

Die freie bzw. offene Verfügbarkeit von digitalisiertem Wissen in Programm- und/oder Datenform führt also auch für den reinen Anwender zu einer Steigerung seiner Möglichkeiten, an wissenszentrierten Prozessen nachhaltig teilzuhaben und damit zu einer Steigerung der „informationalen Chancengleichheit". Denn so kann er hochwertige, preiswerte und vor allem auch von singulären Privatinteressen befreite Software nutzen, um seiner wissenszentrierten Arbeit nachzugehen.[13]

Mit den genannten Punkten eines protestförmigen Willens zum sozialen Wandel und dessen Implikationen sind die letzten fehlenden Elemente im Rahmen der theoretischen Basisdefinition einer sozialen Bewegung untersucht und bestätigt. *Nach der oben genannten, grundlegenden Definition ist das FS/OS-Phänomen nach einer theoretischen Überprüfung als eine soziale Bewegung einzustufen.*

An dieser Stelle zeigt sich trotzt aller Unterschiede zur Arbeiterbewegung des industriellen Entwicklungsmodus noch eine interessante Gemeinsamkeit beider Bewegungen: Beide bedienen sich teilweise der „Mittel des Gegners", um ihre Ziele durchzusetzen. Denn die Kollektivierung der Interessen der Arbeiter gegenüber den Interessen der Arbeitgeber (bzw. des „Kapitals") ist als der Versuch zu werten, die strukturell benachteiligte Position des individuellen Arbeiters durch eine Quasi-Monopolisierung der Lohnaushandlung zu verbessern. Dem Oligopol der Arbeitgeber (wenige bieten vielen die lebensnotwendige und alltagsbestimmende Arbeit an und haben so eine hohe strukturelle Macht) wird ein kollektives Monopol der Arbeitnehmer (die vielen vereinen sich zu einer Gewerkschaft) gegenübergestellt, um bei den wichtigen Lohnaushandlungen ihre Position zu verbessern. Damit versuchte die Arbeiterbewegung auf die damals neuen Strukturen der industrialisierten Erwerbsgesellschaften Einfluss zu nehmen und auf die daraus für sie und den Großteil der Gesellschaft resultierenden Nachteile[14] zu reagieren.

---

[13] Aus diesen Überlegungen jedoch im Rückschluss zu folgern, dass eine Zwangsöffnung aller digitalen Inhalte zu optimaler ökonomischer Effizienz und maximaler Chancengleichheit führt, ist weder theoretisch noch empirisch haltbar. Leider kann eine eingehende Diskussion dieser Frage hier nicht geleistet werden.

[14] (Neo-)Liberale Beobachter sind bis heute der Auffassung, dass Disfunktionalitäten auf den Arbeitsmärkten erst durch nicht-marktgerechte Eingriffe in den Marktmechanismus, z.B. in Form kollekti-

Parallel reagiert hier die FS/OS-Bewegung auf die neuen Strukturen der informationalen Ökonomie. Die Erben der Hacker versuchen heute, ähnlich wie die Arbeiter damals, mit Hilfe ihrer Funktion als zentrale Humanressource des neuen Entwicklungsmodus sich so einzusetzen, dass die sozialen und ökonomischen Nachteile der neuen strukturellen Rahmenbedingungen für sich selbst und andere verhindert bzw. abgemildert werden. Dabei nutzen sie die Waffen des Gegners in gewandelter Form (Arbeiter: Monopolisierung; FS/OS: urheberrechtliche Lizenzen). Soziale Bewegungen scheinen also in Hinsicht auf die Stabilität eines gesellschaftlichen Systems in einem gewissen Umfang eine selbstregulierende Funktion zu haben. „Gerade im Rahmen eines repräsentativ verfassten und eindeutig von politischen Parteien dominierten Institutionensystems kommt dem Spektrum ‚progressiver' Bewegungen eine eminente Rolle als demokratische Produktivkraft zu. Als ein Feld demokratischer Sozialisation, als Modus bürgerschaftlicher Selbsthilfe und advokatorischer Interessenvertretung, als eine Säule kritischer Öffentlichkeit und schließlich als Korrektiv und Innovationspotential für die etablierte Politik tragen sie dazu bei, Demokratie zu beleben und zu festigen" (Rucht 1997, S. 399).

3. Argumentation im Fazit

Die Ausgangsfrage dieses Artikels – ist FS/OS eine soziale Bewegung? – wurde im Ergebnis der Diskussion positiv beantwortet.

Der Verlauf der Argumentation lässt sich wie folgt zusammenfassen: Auf der Basis der genannten Definition einer sozialen Bewegung muss diese mehrere notwendige Bedingungen erfüllen. Neben formalen Kriterien wie Organisationsgrad, geringe Rollendifferenzierung etc. besteht der Kern dieser Bedingungen in der Feststellung einer sozialen Bedeutung der kollektiven, protestförmigen Aktion mit dem Ziel einer gesellschaftlichen Veränderung. Um die soziale Bedeutung von FS/OS und die gesellschaftliche Relevanz der damit verbundenen Aktivitäten zu hinterfragen, wurde die Makroebene des gesamtgesellschaftlichen Zusammenhangs diskutiert. Dieser bezieht sich auf die sozialen und wirtschaftlichen Veränderungen im Rahmen der Entstehung eines informationalen Entwicklungsmodus. Digitalisiertes Wissen wird in diesem Wirtschaftsmodus zu einem Faktor mit neuartiger Bedeutung. Bei dessen Nutzung kommt Software eine zentrale Schnittstellenfunktion zu. In dieser Funktion strukturiert Software ökonomische und soziale Handlungsspielräume der Menschen im Arbeitsalltag. Diese gesellschaftliche Bedeutung von Software stellt die Ausgangsbasis für weiterführende Überlegungen über Art und Zielrichtung der kollektiven Aktionen der FS/OS-Gemeinde dar. Die Mobilisierung

vierter Interessenvertretungen, entstanden sind (vgl. FDP Präsidium 2003, S. 2ff). Diese Grundsatzfrage ist hier nicht zu klären. In diesem Zusammenhang sei jedoch auf die schlichte Tatsache verwiesen, dass der freie Markt, wie bei Adam Smith gedacht, niemals existieren kann. Die gesellschaftliche Ordnung, ihre Institutionen, Normen, Regeln und Gesetze bilden einen notwendigen Bedingungszusammenhang für die kapitalistische Ökonomie (vgl. Polanyi 1978). Dass sich dessen Elemente in der Struktur der Faktormärkte wiederfinden, ist nicht überraschend, kann jedoch in einer demokratisch verfassten Gesellschaftsform nur bedeuten, das den relativ Benachteiligten dieser Strukturen das Recht zur Verbesserung der eigenen Chancen zugesprochen werden muss. Dies bedeutet darüber hinaus, dass es einen „natürlichen Marktzustand" nicht geben kann.

von Teilnehmern zur Herstellung von freier und offener Software unter entsprechenden Lizenzen wird dabei als sozioökonomische Protest- und aktive Widerstandsform interpretiert. Dabei wurden soziale Bezüge auf drei Ebenen festgestellt:

1. Arbeitsalltag und Motivation von Softwareentwicklern
2. Verwertungsmöglichkeiten ökonomischer Akteure im Rahmen von FS/OS
3. Chancengleichheit des Individuums im informationalen Entwicklungsmodus

Diese Ebenen sind miteinander verbunden und begründen die soziale Bedeutung der Aktivitäten der FS/OS-Gemeinde. Zusammen mit der Feststellung weiterer notwendiger Bedingungen ist als Ergebnis dieser theoretischen Überlegungen festzustellen: FS/OS ist als soziale Bewegung einzustufen.

4. Ausblick

Mit der hier vorgenommenen theoretischen Prüfung ist zwar ein erster Schritt zu einem besseren Verständnis der sozialen Dimension von FS/OS getan. Eine Vielzahl an Aspekten konnte jedoch nur angeschnitten bzw. festgestellt werden oder musste aus Umfangsgründen gänzlich ausgespart bleiben. Dabei bleiben mehr Fragen offen als beantwortet werden konnten. Auf beiden wissenschaftlichen Gebieten der Informatik und Sozialwissenschaften existieren noch diverse Forschungslücken, die in diesem Zusammenhang zu schließen sind. Einige sollen hier noch kurz erwähnt werden.

Die gegenwärtige sozialwissenschaftliche Bewegungsforschung geht über die hier genutzte Basisdefinition weit hinaus. Als erster aufgeklärter Ansatz gilt die Theorie der *relativen Deprivation*, die die Entstehung sozialer Bewegungen aus der „empfundenen Diskrepanz zwischen der eigenen Lage und der Situation vergleichbarer Bezugsgruppen" (Schaffhauser 1997, S. 10) ableitet. Der *Ressourcenmobilisierungsansatz* erweitert diese Sicht und betrachtet soziale Bewegungen vor allem in Hinsicht auf ihre öffentlichkeitswirksamen Aktivitäten. Die Strukturen der modernen Mediengesellschaft müssen von den Bewegungen zur Mobilisierung ihrer Teilnehmer benutzt werden und diese benötigen dazu Ressourcen. Soziale Bewegungen sind daher auch über ihre Kommunikations- und Ressourcenmobilisierungsmuster im Kontext der Massenmedien zu analysieren und zu verstehen. Die Chancen und Gegebenheiten zur effektiven Nutzung dieser Ressourcen werden durch das Konzept der *politischen Gelegenheitsstrukturen* ergänzt, das vor allem die historisch-gesellschaftspolitischen Fenster für das erfolgreiche Entstehen sozialer Bewegungen berücksichtigt. Das eher konstruktivistische Framing-Konzept und die neueren Untersuchungen der Biographieforschung unterstreichen zunehmend die Bedeutung der gesellschaftlichen Mikroebene für die Untersuchung sozialer Bewegungen (vgl. Schaffhauser 1997; Gajdukowa 2002; Kolb 2002; Neidhardt 1994). Diese Aufzählung ließe sich lange fortsetzen. Die Anwendung dieser Erkenntnisse auf FS/OS wäre zwar sehr interessant, würde jedoch den gegebenen Rahmen bei weitem sprengen und stellt ein fruchtbares Feld für weitere Untersuchungen dar.

Auch aus anderen Perspektiven sind noch viele Fragen offen. Neben der Konkretisierung der ökonomischen Implikationen eines informationalen Entwicklungs-

modus und den daraus folgenden Schlüssen für Betriebs- und Volkswirtschaft betreffen diese Fragen vor allem den sinnvollen Einsatz und die Zukunft von FS/OS. Welche Branchen sind für den Einsatz des FS/OS-Paradigmas prädestiniert? Lässt sich die Logik offener Architekturen sinnvoll auf nicht-technologische Bereiche übertragen? Welche soziologische Rolle kommt den FS/OS-Führungsfiguren (Stallman, Torvalds, Raymond etc.) und ihrem jeweiligen Hintergrund zu? Werden die juristischen Konstruktionen der freien und offenen Lizenzen auf Dauer den Angriffen der proprietären Konkurrenz standhalten?

Vor allem bleibt jedoch die Beurteilung der Bedeutung der FS/OS-Bewegung im Vergleich mit anderen sozialen Bewegungen, wie z.B. der Arbeiterbewegung, der Friedensbewegung oder der Umweltbewegung eine Frage, die nur aus einer gewissen Distanz seriös beantwortet werden kann. Denn die wissens- und informationszentrierte Entwicklung der Gesellschaft, die hier als informationaler Entwicklungsmodus bezeichnet wurde, steht erst an ihrem Anfang. Nimmt man die Geschichte der industriellen Entwicklung zum Maßstab, dann sind noch einige unvorhersehbare Innovationen mit diversen sozioökonomischen Seiteneffekten und Turbulenzen zu erwarten. Unsere Gesellschaften werden in 50 Jahren anders aussehen, als wir uns dies heute vorstellen können. Welche Rolle FS/OS bis dahin spielen wird, bleibt abzuwarten.

5. Literatur

Altvater, Elmar / Mahnkopf, Birgit (2000): *New Economy – nichts Neues unter dem Mond?* In WSI Mitteilungen 12/2000, S. 770–777

Arthur, W. Brian (1994): *Increasing returns and path dependence in the economy*, Michigan.

Arthur, W. Brian (1999a): *Complexity and the Economy*, in: Science 284, S. 107–109.

Arthur, W. Brian (1999b): *The End of Certainty in Economics*, in: Aerts/Broekaert/Mathijs (Hg.): Einstein meets Magritte: An Interdisciplinary Reflection on Science, Nature, Art, Human Action and Society. Vol. 1, Dordrecht.

Castells, Manuel (2000): *The rise of the network society*, Oxford u. a.

Christensen, Clayton M. (2000): *The Innovator´s Dilemma*, Boston.

FDP Präsidium (2003): *Liberale Vorschläge zur Beseitigung der Arbeitsmarktmisere*, Drucksache 15/15. Wahlperiode, Berlin 11. 3. 2003,
online http://mdb.liberale.de/fraktion/dateien/initiativen/Antrag_Arbeitsmarktpolitik12032003.pdf.

Gajdukowa, Katharina (2002): *Paradigmenwechsel in der Forschung zu sozialen Bewegungen*, Forum Qualitative Sozialforschung Vol.3, No.4 2002.

Gordon, Robert J. (2000): *Does the New Economy measure up to the great inventions of the past?* In: Journal of economic perspectives 14/2000, S. 49–74.

Grahl, John; Paul Teague (2000): *The Régulation School, the employment relation and financialization*, in: Economy and Society Vol. 29 No. 1 2000, S. 160–178.

Grassmuck, Volker (2002): *Freie Software. Zwischen Privat- und Gemeineigentum*, Bonn.

Greve, Georg C.F. (2003): *Free Software in Europe*, FSF-Europe. dowload http://fsfeurope.org/documents/eur5greve.html.

Gröndahl, Boris (2002): *The tradegy of the anticommons*, in: Wissen und Eigentum im digitalen Zeitalter, Prokla 1/2002.

Hübner, Kurt (1989): *Theorie der Regulation – Eine kritische Rekonstruktion eines neuen Ansatzes der politischen Ökonomie*, Berlin.

Hofmann, Jan (2002): *Free Software, big business ? Open-Source-Programme erobern Wirtschaft und öffentlichen Sektor*, Deutsche Bank Research Nr. 32/2002, online http://www.dbresearch.de.

Jargon File 4.0.0 (div.): online http://www.jargon.8hz.com/jargon_toc.html.

Kocka, Jürgen (2001): *Thesen zur Geschichte und Zukunft der Arbeit*, in: Aus Politik und Zeitgeschichte 21/2001.

Kolb, Felix (2002): *Soziale Bewegungen und politischer Wandel*, Bonn.

Lampert, Heinz (1998): *Lehrbuch der Sozialpolitik*, Berlin u. a.

Lessig, Lawrence (2001): *The Future of Ideas: The Fate of Commons in a connected World*, New York.

Levy, Stephen (1984): *Hackers*, Penguin Press, online http://www.stanford.edu/group/mmdd/SiliconValley/Levy/Hackers.1984.book/contents.html.

McFadden, Daniel (1998): *Rationality for Economists?* Working Paper am Santa Fé Institute, online http://www.santafe.edu.

Müller-Jentsch, Walter (1985): *Soziologie der Industriellen Beziehungen*, Frankfurt/M.

Neidhardt, Friedhelm (1994): *Öffentlichkeit, öffentliche Meinung, soziale Bewegungen*, in: Kölner Zeitschrift für Soziologie und Sozialpsychologie, Sonderheft 34, S. 7–41

Ostrom, Elinor (1991): *Governing the Commons: The Evolution of Institutions for Collective Action*.

Polanyi, Karl (1978): *The Great Transformation*, Wien.

Raschke, Joachim (1988): *Soziale Bewegungen. Ein historisch-systematischer Grundriß*, 2. Aufl. der Studienausgabe, Frankfurt/M.

Raymond, Eric S. (2001): *The Cathedral* & *The Basar. Revised Edition*, O'Reilly, deutsche Fassung online http://www.linux-magazin.de/Artikel/ausgabe/1997/08/Basar/basar.html.

Rucht, Dieter (1997): *Soziale Bewegungen als demokratische Produktivkraft*, in: Klein / Schmalz-Bruns(Hg.): Politische Beteiligung und Bürgerengagement in Deutschland, Bonn.

Schaffhauser, Roman (1997): *Öffentlichkeit und soziale Bewegungen*, Sociology in Switzerland/Online Publications, online http://socio.ch/movpar/t_rschaff1.htm.

Stallman, Richard (1992): *Why Software Should Be Free*, online http://www.fsf.org/philosophy/shouldbefree.html.

Stallman, Richard (1999): *The GNU Project*, in: Open Sources : Voices from the Open Source Revolution, online http://www.oreilly.com/catalog/opensources/book/stallman.html.

Weber, Steven (2000): *The Political Economy of Open Source Software*, BRIE Working Paper 140, University of California, Berkeley,
online http://repositories.cdlib.org/brie/BRIEWP140.

Zimmermann, Thomas (2002): *New Economy – veränderte sozio-ökonomische Rahmenbedingungen der Arbeit in der informationalen Wirtschaft*, Diplomarbeit am Institut für Sozialwissenschaften der Philosophische Fakultät III der Humboldt-Universität zu Berlin,
online http://www.kmgne.de/download/publ_neweconomy.htm.

Philosophische Grundlagen und mögliche Entwicklungen der Open-Source- und Free-Software-Bewegung

KARSTEN WEBER

Wie alles in der Welt der Informations- und Kommunikationstechnologie und der Informationsgesellschaften sind die Open-Source- und Free-Software-Bewegung[1] vergleichsweise jung: Ihre historischen Wurzeln liegen in den 1980er Jahren. Nichtsdestotrotz haben sie in kurzer Zeit erhebliche Wirkungen im Bereich der Softwareentwicklung und auf dem Markt für Software erzielt. Gezeigt wurde, dass es möglich ist, hochwertige Programme und ein komplettes Betriebssystem inklusive zahlreicher Anwendungen durch die hoch verteilte Tätigkeit freiwillig zusammenarbeitender Menschen zu entwickeln. Auf dieser Basis ist eine Alternative zu den proprietären Betriebssystemen und Office-Produkten des Quasi-Monopolisten Microsoft entstanden und neue Bewegung in die Softwaremärkte hineingetragen worden.

Die Historie non-proprietärer Software ebenso wie Technik, Einsatzmöglichkeiten und Bedeutung für den Softwaremarkt werden in anderen Beiträgen des vorliegenden Bandes kompetent diskutiert; im vorliegenden Text soll auf philosophische Grundlagen quelloffener Software eingegangen und versucht werden, zukünftige Entwicklungen vorwegzunehmen. Allerdings müssen diesem Unterfangen einige Anmerkungen vorangestellt werden, um Fehlinterpretationen zu vermeiden:

1. Insbesondere die Diskussion der philosophischen Grundlagen muss als Rekonstruktion verstanden werden. Es ist plausibel, die quelloffener Software zu Grunde liegenden Ideen mit libertären sowie zum Teil auch kommunitaristischen Positionen der politischen und Sozialphilosophie zu identifizieren, doch muss dies nicht heißen, dass alle Proponenten non-proprietärer Software dem zustimmen würden oder sich dieser Hintergründe bewusst wären. Der hier gewählte Zugang ist systematisch orientiert und greift die historische Entwicklung des Libertarismus nicht auf; stattdessen sollen dessen zentrale Ideen und ihre Verbindung zu quelloffener Software vorgestellt werden.

2. Die Erörterung möglicher Entwicklungen hat den Charakter unsicherer Prognosen. Trotz allem Enthusiasmus der Unterstützer quelloffener Software ist diese in ihren allgemein sozialen Auswirkungen immer noch als marginal zu bezeichnen; Open-Source- und Free-Software-Bewegung als soziale Bewegungen können sich hinsichtlich ihrer Bedeutung zurzeit noch nicht mit Umweltschutz-, Menschenrechts- oder Antiglobalisierungsbewegungen messen. Ob Open Source und Free Software untereinander vergleichbar sind, kann ebenfalls infrage gestellt werden –

[1] Andere Bezeichnungen sind „quelloffene", „freie" oder „libre software" (vgl. Robles u.a. 2001, S. 1); Aaron M. Renn (1998) schlägt die Bezeichnung „non-proprietäre Software" vor. Im Folgenden werden „quelloffen", „frei" und „non-proprietär" synonym verwendet.

Richard M. Stallman (2001) wirft Open Source ja vor, in erster Linie technologische und nicht soziale Aspekte zu beachten. Wie sie sich entwickeln werden, ist offen.

3. Stellungnahmen zu non-proprietärer Software befinden sich stets in der Gefahr, zu persönlichen Bekenntnissen für oder wider solche Software und der dahinter liegenden Weltanschauungen zu mutieren. Zumindest liegt dieser Eindruck nahe, wenn man die Diskussionen um quelloffene Software aufmerksam verfolgt. Die Form der Auseinandersetzung verstellt oft den Blick auf einige Aspekte non-proprietärer Software, die gerade hinsichtlich ihrer sozialen Bedeutung bedenkenswert und bedenklich erscheinen.

Der Objektivität von Stellungnahmen zu quelloffener Software sind Grenzen gesetzt, sie ist oft von Loyalität oder Gegnerschaft überschattet und Diskussionen zu diesem Thema sind selten emotionslos. Dies gilt es, bei der Lektüre *aller* Texte zu quelloffener Software *immer* zu beachten; zudem ist das bereits eine der Konsequenzen, die aus der Entwicklung non-proprietärer Software gezogen werden sollten: Sachlichkeit, Distanz und größtmögliche Objektivität sind Werte, die in der Diskussion um quelloffene Software häufig erst wieder neu erreicht werden müssen. Das gilt sowohl für Proponenten als auch für Opponenten quelloffener Software.

1. Libertäre Grundlagen

Wie andere soziale Bewegungen auch – man denke hier bspw. an die bundesdeutsche Umweltbewegung der 1980er Jahre – sind Open-Source-Gemeinde und Free-Software-Bewegung erheblich durch eine starke Personalisierung von Konflikten gekennzeichnet. In Bill Gates als wichtigstem Vertreter von Microsoft bzw. eines bestimmten Geschäftsmodells bündeln sich die Antipathien gegenüber proprietärer Software; Richard M. Stallman, Eric S. Raymond und insbesondere Linus Torvalds sind hingegen die Lichtgestalten der sozialen Bewegung rund um quelloffene Software. Die Tendenz zur Personalisierung findet eine ihrer Ursachen sicherlich in der Persönlichkeit der „Frontmänner“[2]: Stallman steht für die *Free Software Foundation*, Raymond repräsentiert Open Source und Linus Torvalds hat LINUX ins Leben gerufen. Alle drei besitzen Sendungsbewusstsein, ihre Schriften und Bücher geben davon ein deutliches Zeugnis – übrigens gilt das auch für Bill Gates. Philosophisch und weltanschaulich äußern sich öffentlich vor allem Stallman und Raymond, auch wenn Free Software und Open Source in einer Art von Konkurrenz hinsichtlich der zu Grunde liegenden Weltanschauungen und der sozialen Bedeutung „ihrer“ Bewegungen stehen. Beide bezeichnen sich selbst als „libertär“; um also die Hintergrundphilosophie non-proprietärer Software zu verstehen, kommt man nicht umhin, einen Blick auf libertäres Gedankengut zu werfen.[3]

---

[2] Non-proprietäre Software ist fast vollständig Männersache (Hertel u.a. 2003, S. 18; Robles u.a. 2001, S. 26).

[3] Wenn hier Stallman und Raymond als „Kronzeugen“ verschiedener Positionen angeführt werden, so soll dies nicht den beschriebenen Hang zur Personalisierung nachvollziehen, sondern ist der Kürze der Formulierungen geschuldet: Open Source als auch Free Software bestehen selbstverständlich nicht nur aus einer Person.

1.1 Staatsverständnis und Menschenbild[4]

Der Libertarismus ist weit älter als das Internet, Open Source oder Free Software.[5] Seine Wurzeln liegen im radikal-liberalen Denken und in anarchistischen Ideen, in denen (staatliche) Autorität grundsätzlich abgelehnt wird. Bei Robert Nozick als einem der wichtigsten Vertreter zeitgenössischer libertärer Positionen in der politischen Philosophie ist der Staat auf elementare Aufgaben beschränkt: „[…] a minimal state, limited to the narrow functions of protection against force, theft, fraud, enforcement of contracts, and so on, is justified" (Nozick 1974, S. IX). Vom Staat, den wir empirisch kennen, bleiben nur die Strafverfolgungsbehörden übrig, das Militär und ein Rumpffinanzapparat, der die Mittel für die zwei erstgenannten Institutionen eintreibt. Dieser Minimalstaat ist ein „Nachtwächterstaat" (vgl. Nozick 1974, S. 26), der dafür sorgt, dass das Miteinander der Menschen nach Recht und Gesetz abläuft, der innere und äußere Bedrohungen abwehrt, sich aber ansonsten aus dem öffentlichen und privaten Leben seiner Bürger völlig heraushält.

Dass nur der Minimalstaat moralisch und politisch legitimiert ist, wird von Libertären durch die Annahme eines vorstaatlichen Zustandes und der Entstehung des Staates aus diesem verdeutlicht. Dabei ist der so genannte „Naturzustand", auf den bspw. Nozick rekurriert, dem von John Locke im „Second Treatise" im 2. Kapitel „Of the State of Nature" (2000) beschriebenen Zustand ähnlich und nicht dem „Krieg aller gegen alle", wie er von Thomas Hobbes im „Leviathan" (1992; vgl. Baurmann 1994, S. 105) angeführt wird. Das bedeutet, dass der Naturzustand als vergleichsweise friedlich verstanden wird – der „[…] beste […] staatsfreie […] Zustand, auf den man vernünftigerweise hoffen kann" (Nozick ohne Jahr, S. 20). Locke (§6) formuliert, dass jedem Menschen bestimmte Rechte natürlicherweise zukämen: „And Reason, which is that Law, teaches all Mankind, who will but consult it, that being all equal and independent, no one ought to harm another in his Life, Health, Liberty, or Possessions." Rechte müssen aber auch geschützt werden können. Im Naturzustand, vor jeder Entstehung eines Gemeinwesens oder Staats, müssen und dürfen die Menschen das Recht in die eigene Hand nehmen (§7): „[…] every one has a right to punish the transgressors of that Law to such a Degree, as may hinder its Violation." Da Selbstjustiz aber zu Problemen führen kann – vor allem zu übertriebenen und falschen Strafen, die Racheakte nach sich ziehen können –, muss hier eine andere Lösung gefunden werden: der Staat als Agentur, die die Rechte der Menschen schützt. Diese Agentur kommt dadurch zustande, dass sich Menschen zunächst zu „Schutzbündnissen" zusammenfinden, diese dann privatwirtschaftlich organisiert werden und zuletzt jenes Schutzbündnis, das in der Konkurrenz mit anderen am erfolgreichsten war, auf einem Territorium das eigene Gewaltmonopol durchsetzt. Damit ist der libertäre Minimalstaat entstanden (Nozick 1974, S. 113 ff.).

---

[4] Abschnitt 1.1 und 1.2 basieren auf Teilen meiner Habilitation „Informationelle Grundversorgung und Eingriffsfreiheit. Der Zugang zu Informationen aus der Perspektive politischer Philosophie", die am 11. Juni 2003 an der Europa-Universität Viadrina Frankfurt (Oder) zur Begutachtung angenommen wurde.

[5] Eine Übersicht zum libertären Gedankengut und seiner historischen Entwicklung bietet Peter Mühlbauer (TP 2000) in seiner Artikelserie im Online-Magazin „Telepolis".

Der Minimalstaat ist aus libertärer Sicht moralisch legitimiert, weil er auf freien Übereinkünften freier Menschen basiert. Deren Ziel war nicht, so die Annahme, einen Staat zu gründen, sondern Maßnahmen zu ergreifen, um sich jeweils selbst zu schützen. Der Staat als Minimalstaat ist dabei gleichsam als Randerscheinung entstanden – durch das Wirken der so genannten „Unsichtbaren Hand" (Nozick 1974, S. 18 ff.). Da der einzige Zweck des Staates ist, die Rechte der Bürger zu schützen, darf er keine Schritte unternehmen, in die Rechte einer Person einzugreifen, solange nicht die Rechte anderer Personen beeinträchtigt werden. All das, was wir heute empirisch als staatliche Aufgaben (an-)sehen, ist aus libertärer Sicht moralisch und politisch illegitim. Dies gilt insbesondere für alle Umverteilungsmaßnahmen, mit denen ökonomische Unterschiede zwischen Bürgern eines Staates ausgeglichen werden sollen.

1.2 Eigentum

Ausgangspunkt für diese Sicht ist das Konzept des Selbsteigentums. Es wird der Anspruch erhoben, dass jede Person sich selbst besitzt und vollständig über den eigenen Körper bestimmen kann bzw. niemand ein Recht besitzt, in dieses Selbsteigentum einzugreifen (Nozick 1974, S. 26 ff.; Wolff 1991, S. 7 f.; vgl. Gorr SPP 1995; Morimura ARSP 1993). Angelehnt ist das an Kants zweiter Ableitung des kategorischen Imperativs (GMS, S. 429): „Handle so, dass du die Menschheit, sowohl in deiner Person als in der Person eines jeden anderen, jederzeit zugleich als Zweck, niemals bloß als Mittel brauchst."

Wenn man das Selbsteigentum nicht anerkennen würde, wäre es möglich, zur Steigerung des Wohls der größten Zahl oder zum Zwecke der Herstellung von Gleichheit, dass andere Personen oder der Staat auf den Körper einer Person Zugriff nehmen dürften. Wolff (1991, S. 7f.) macht das mit folgendem Beispiel klar: Man nehme an, dass es eine zu 100% sichere Methode der Augentransplantation gäbe; es wäre also möglich, zwischen Personen Augen zu tauschen, sodass diese im neuen Körper ohne Einschränkungen funktionieren würden. Ohne Zweifel könnte man nun das Wohl von Blinden steigern, wenn ihnen wenigstens ein sehendes Auge implantiert würde. Dafür aber benötigte man Spender. Was aber, wenn mehr Blinde als freiwillige Spender existierten? Soll Zwang ausgeübt werden? Nozick (1974, S. 32) beantwortet dies mit Kant strikt negativ, denn es bedeutete, einen Menschen zum Mittel zu degradieren.

Eigentum bedeutet für Libertäre also, andere von der Nutzung von Gütern auszuschließen oder doch ihre Möglichkeiten zur Nutzung einzuschränken bzw. dem Eigentümer das Recht einzuräumen, frei über die Verwendungsweise zu entscheiden. Nozick (1974, S. 174 ff.) geht in seiner Begründung gerechten Eigentums auf Locke (kritisch Ryan SPP 1994, S. 247f.) und dem eigenen Argument des Selbsteigentums zurück. Locke formuliert in § 27 und den darauf folgenden Paragrafen des „Second Treatise", dass zunächst die Arbeit ihres Körpers jeder Person selbst gehöre. Arbeit jedoch wird an etwas verrichtet. Dadurch, dass nun die Arbeit und die Dinge der Welt gleichsam vermischt würden, erwerbe die arbeitende Person einen Eigentumsanspruch an dem Bearbeiteten. Dabei sei es nicht notwendig (§28), dass andere Personen der Aneignung zustimmten, sofern sich nur jene materiellen Din-

ge, die durch die Arbeit angeeignet werden sollen, nicht schon im Besitz einer anderen Person befänden. Die Dinge, die auf solche Weise in Besitz genommen werden können, sind im Grunde alle materiellen Körper außer Menschen, da diese Eigentum an sich selbst besitzen. Nozick fügt eine Bedingung hinzu, die erfüllt sein müsse, damit die Aneignung gerecht sei (1974, S. 178), die unter der Bezeichnung „Lockes Proviso" bekannt ist (vgl. Jasay A&K 1999, S. 161 ff.; Leist 2003, S. 26; Mack SPP 1995): Durch die Inbesitznahme dürfe niemand schlechter gestellt werden als vorher. Das heißt, dass bspw. im Fall einer Landnahme jene, die vorher ihren Lebensunterhalt durch Sammeln von Früchten auf diesem Land bestritten haben, durch irgendwelche kompensatorischen Maßnahmen entschädigt werden müssen. So könnten sie nun als Beschäftigte des Landbesitzers arbeiten (Beispiel aus Kymlicka 1997, S. 114 ff.; vgl. Kern 2001, S. 200f.). Würden sie dadurch materiell nicht schlechter gestellt als vor der Landnahme, so wäre diese nach Nozick gerecht.[6] Lockes Proviso geht allerdings über Nozicks Konzeption hinaus, denn dort gilt, dass die Aneignung eines Gutes nur dann legitim ist, wenn allen anderen Menschen von diesem Gut genügend übrig bleibt (Kern 2001, S. 200).

1.3 Anwendung und Rekonstruktion

Interessanterweise ziehen Stallman und Raymond aus ihrem Bekenntnis zu libertären Ideen unterschiedliche Konsequenzen. Stallman ist der Ansicht, dass Eigentum an Informationen oder Wissen – und Software gehört für ihn dazu – illegitim sei: „Information wants to be free" (vgl. Stallman 1992). Dies könnte so rekonstruiert werden, dass er mit Locke der Ansicht ist, dass die Inbesitznahme von Informationen und damit die alleinige Verfügungsgewalt über sie dazu führen müsse, dass in der Folge für alle anderen Menschen ein eklatanter Mangel entstünde. Tatsächlich kann man dieses Argumentationsmuster bei einer Reihe von Autoren finden, bspw. bei Helmut F. Spinner (z.B. Spinner 1994; Spinner 2002) oder Volker Grassmuck (z.B. Grassmuck 2002). Beide argumentieren, dass die gemeinsame Nutzung von Informationen und Wissen einen für die Allgemeinheit größeren Nutzen erzeuge als die exklusive Verwendung durch einzelne Personen oder Institutionen. Damit lehnen sie sich am so genannten CUDOS-Modell des Wissenschaftskommunismus an, das von Robert K. Merton (1985, S. 86 ff.) entwickelt wurde. Im Umkehrschluss führt die exklusive Nutzung zu einem Schaden oder Mangel.

Man kann aber auch an einem anderen Punkt einhaken, um freie Software zu legitimieren (vgl. Raymond 2000a, S. 8 ff.). Denn im libertären Gedankengebäude nimmt Eigentum einen hohen Stellenwert in der Werteskala ein; Eigentumsrechte werden bei Nozick und anderen Libertären absolut gesetzt – wurde ein Gut moralisch legitim erworben, entweder durch Erstaneignung eines herrenlosen Guts oder durch fairen Tausch, darf niemand in dieses Eigentumsrecht eingreifen. Erwirbt also eine Person ein Computerprogramm, so erwirbt diese Person das Recht, alles damit zu tun, was sie will. Dies könnte so gedeutet werden, dass jeder Mensch, der

[6] Nozick (174, S. 141, 181f.) nimmt auch Stellung zur Frage von Patenten; sie sind nach seiner Meinung unter bestimmten Bedingungen durchaus moralisch legitimiert. Das widerspricht wiederum der im Bereich non-proprietärer Software weit verbreiteten Ablehnung von Softwarepatenten und vergleichbaren Regelungen für geistiges Eigentum.

Software moralisch legitim erworben hat, diese verkaufen, verschenken, durchschauen und verändern darf. Hier unterscheiden sich Stallman und Raymond in ihrer Interpretation dessen, was sie „libertär" nennen. Raymond (2000a, S. 3 ff.) lässt durchaus zu, dass Software im üblichen Sinne ein handelbares Gut ist, dessen Nutzung durch Lizenzverträge eingeschränkt werden kann – Software kann also auch proprietär sein. Dies wiederum kann daraus abgeleitet werden, dass Vertragspartner völlig frei sind, die Bedingungen eines Tauschs – bspw. Geld gegen Software – auszuhandeln. Bestandteil solcher Bedingungen kann eben auch sein, nicht auf den Quellcode zugreifen oder das Programm nicht weitergeben zu dürfen. Nur ist Raymond der Ansicht, dass dies letztlich ineffizient sei und nicht den tatsächlichen Bedingungen der Softwareerstellung entspräche (vgl. Raymond 2000b, S. 4 ff.); die Entscheidung für freie Software sieht er in ihren Vorteilen gegenüber proprietärer Software begründet (vgl. O'Reilly TP 1999), nicht in irgendwelchen letztinstanzlichen Grundsatzentscheidungen. Stallman und andere Autoren (z.B. Renn 1998) hingegen argumentieren, dass Software grundsätzlich frei sein solle, weil der freie – nicht notwendig kostenlose – Austausch von Software ein Akt der Solidarität und Kooperation sei, ohne den Gesellschaften nicht wirklich funktionierten (Stallman TP 1999).

Stallman betont immer wieder, dass „frei" nicht „kostenlos" bedeute (Stallman 2003). Den Schritt, alle Güter zu sozialisieren, vollzieht er ausdrücklich nicht. Dies bleibt anderen Autoren überlassen, die in quelloffener Software einen Beitrag hin zu einem Marxismus bzw. Kommunismus sehen (bspw. Söderberg FM 2002).[7] Bei allen Unterschieden kann man Raymond und Stallman jedoch darin einig sehen, dass sie gesetzliche Regelungen wie Urheberrechte oder Softwarepatente ablehnen, weil sie moralisch illegitime Eingriffe des Staates in die Eigentumsrechte der Bürger darstellen (Stallman Upgrade 2001; Stallman 2002). Dies steht allerdings im klaren Gegensatz zu libertären Autoren wie Nozick.[8]

2. Kommunitaristische Aspekte

Obwohl bei wichtigen Vertretern der Open-Source- und Free-Software-Bewegung libertäre Elemente im Vordergrund stehen und damit vor allem die Rechte der je einzelnen Person ins Blickfeld gerückt werden, finden sich auch andere Elemente sozialphilosophischer Positionen wieder. Anarchistische, marxistische und kommunistische Aspekte waren bereits angesprochen worden. Im Folgenden soll aber ein Gesichtspunkt thematisiert werden, der in der Diskussion der politischen und Sozialphilosophie den libertären Gedanken diametral gegenüber steht. Libertarismus und Anarchismus lassen sich durchaus vereinigen – der Anarcho-Syndikalismus Noam Chomskys könnte als Beispiel genannt werden. Dass sich aber in der Gedankenwelt der Vertreter non-proprietärer Software kommunitaristische und libertäre Aspekte berühren und zusammengehen, ist ungewöhnlich. Um dies besser verste-

[7] Siehe bspw. auch viele Texte zu den Oekonux-Konferenzen, in denen sich anarchistische, libertäre und marxistische Ideen mischen (http://www.oekonux-konferenz.de/dokumentation/texte/index.html, Stand 07/2003).

[8] Vgl. Fußnote 6.

hen zu können, müssen zunächst einige kommunitaristische Grundideen verdeutlicht werden.

2.1 Ursprünge[9]

Die kommunitaristische Debatte ist vergleichsweise jung; sie gewann ihren Einfluss auf die Diskussionen der politischen Philosophie und die alltägliche Politik erst in den 1980er Jahren, ausgehend vor allem von den USA und Kanada (vgl. Reese-Schäfer 1994, S. 7). Allerdings wird von Kommunitaristen – in der Literatur findet sich auch die Bezeichnung „Kommunitarier" – nicht im luftleeren Raum oder ahistorisch gedacht: Sie greifen durchaus auf klassische Autoren zurück – Martha C. Nussbaum (1999; vgl. Nussbaum 2000, S. 89 ff.; Sturma 2000) bspw. gründet ihre politische Philosophie auf Aristoteles, andere machen Anleihen bei Rousseau, Hegel oder Tocqueville (Forst 1995, S. 181f.; vgl. Rorty 1992, S. 236f.) Gleichzeitig greifen Kommunitaristen in ihren Texten sehr häufig, wie Reese-Schäfer (1994, S. 8) bemerkt, auf „[…] den Appell an eingelebte Verhaltensweisen und Traditionen – vor allem an Traditionen des Protests und der Reform wie die amerikanische Bürgerrechtsbewegung" zurück. Sie sehen ihre Aufgabe in erster Linie darin, Kritik an bestehenden politischen Verhältnissen zu äußern: „Communitarians clearly detest much of our contemporary society." (Frohnen 1996, S. 59). Insoweit ist ihr Anliegen praktisch ausgerichtet; wo sie theoretisch arbeiten, kritisieren sie meist liberale und libertäre Theorien, jedoch immer mit Bezug auf konkrete Beispiele, häufig auch mit empirisch-sozialwissenschaftlicher Herangehensweise (vgl. Erben 2000, S. 134; Rössler 1994, S. 74; Walzer 1992).

2.2 Kritik an der libertären Konzeption der Person

Dem libertären Denken liegt ein bestimmtes Menschenbild zu Grunde. Amitai Etzioni (1990, S. 10) charakterisiert es mit folgenden Worten – wobei er „libertär" im folgenden Zitat mit „neoklassisch" übersetzt:

> At the core of the neoclassical paradigm is the assumption that freestanding individuals are the decision-making unit, the actor. This is much more than a working hypothesis; it is an article of faith grounded in a deep commitment to the value of liberty. Neoclassicists argue that if one assumes that the preferences of individuals can be manipulated or changed by social forces, one undermines the foundation of liberty – the notion that each individual is able to render decisions on his own.

Diese Charakterisierung der völlig autonomen Person wird besonders deutlich bei Nozick als paradigmatischen Vertreter des Libertarismus. Taylor (1996, S. 202; vgl. Mulhall und Swift 2002, S. 111 f.) beschreibt die libertäre[10] Sicht mit einer Organ-Metapher: „Die Anhänger dieser Denkrichtung sind anzunehmen geneigt, wir hätten ein Selbst in der gleichen Weise, in der wir Herz und Leber haben, nämlich als ein interpretationsfrei gegebenes Etwas." Doch Kommunitaristen wie Etzioni (1990, S. 10) sind der Meinung, dass diese Konzeption der Person sowohl normativ

[9] Die beiden folgenden Abschnitte zu den Grundlagen des Kommunitarismus basieren wiederum auf Teilen meiner Habilitation. Vgl. Fußnote 4.

[10] Der Einfachheit halber wird hier die kommunitaristische Gleichsetzung von Libertarismus und Liberalismus übernommen, ohne damit zu implizieren, dass dies adäquat wäre.

als auch deskriptiv nicht haltbar sei und verweisen auf sozialwissenschaftliche und psychologische Erkenntnisse, die zeigten, dass gerade jene Individuen, die in einen stabilen gesellschaftlichen Kontext mit funktionierenden Normen und Werten eingebunden sind, weitaus besser ihr Leben gestalten und die dabei notwendig werdenden Entscheidungen treffen könnten (ebd.). Außerdem formuliert er, dass eine Ansammlung von Menschen – von Gesellschaft wäre in diesem Fall eher nicht mehr zu sprechen –, die über keine gemeinsamen Werte und Normen mehr verfüge, am stärksten den Gefahren von diktatorischen und totalitären Tendenzen ausgesetzt sei. Freiheit, so seine These, ließe sich am besten in einer intakten Gesellschaft sichern, da es durch die geteilten Normen und Werte Orientierungspunkte für das eigene Handeln und für die Bewertung des Handelns anderer gäbe (ebd.).

Doch diese Freiheit und Identität ist eben nicht mehr diejenige der Libertären. Die freien Menschen Etzionis bzw. der Kommunitaristen sind jene, die in eine Gesellschaft eingebettet, die darin aufgewachsen und durch sie geprägt sind. Kommunitaristen sind überzeugt, dass „the identity of the autonomous, self determining individual requires a social matrix, one for instance which through a series of practices recognizes the right to autonomous decision and which calls for the individual having a voice in deliberation about public action." (Taylor 1992, S. 209). Das rationale und freie Individuum sprieße nicht einfach aus dem Boden wie Pilze nach dem Regen, sondern es bedürfe der Gemeinschaft, ihres Schutzes, ihrer Werte und ihrer Normen, um überhaupt autonom werden zu können. Kymlicka (1997, S. 181) fasst die Unterschiede zwischen Liberalen und Kommunitaristen so zusammen: „In liberaler Sicht verlangt die Frage des guten Lebens ein Urteil darüber, was für ein Mensch wir werden möchten; in kommunitaristischer Sicht müssen wir herausfinden, wer wir schon sind." Für Kommunitaristen ist das liberale Bild von Personen jenes vom „frei schwebenden Ich" bzw. „ungebundenen Selbst", das ohne Bindungen seine Lebenspläne entwickelt (vgl. Sandel 1995). Dem stellen sie die immer schon gegebene Einbettung der Person in soziale Praxen, Normen und Werte und Lebensziele entgegen.

Dies hat Konsequenzen für die normativen Ansprüche an den Staat; Kommunitaristen haben weiter gehende Ansprüche als Libertäre. Er soll Verteilungsgerechtigkeit herstellen und ist deshalb berechtigt, Güter umzuverteilen und somit in die Eigentumsrechte der Bürger einzugreifen. Umverteilung soll durch Transferleistungen an sozial Schwächere und durch die Finanzierung öffentlicher Dienste erreicht werden, sofern dies dazu beiträgt, das Gemeinwohl zu erhalten und die geteilten Normen und Werte zu befördern. Der Staat darf und soll Mittel, die von den Mitgliedern der Gesellschaft aufgebracht werden müssen, dazu nutzen, Museen, Theater oder auch öffentlich finanzierte Forschung bzw. allgemeine Dienstleistungen kultureller Art zu unterstützen. Solche Einrichtungen sollen unabhängig vom Markt, und damit einer Nachfrage, existieren. Kommunitaristen argumentieren „Gegen die Tyrannei des Marktes" (Bellah u.a. 1994), weil dieser ihrer Ansicht nach die natürlichen Lebensgrundlagen der Menschen zerstöre und zudem auf lange oder sogar schon mittlere Sicht die Freiheitsräume der Menschen viel stärker einenge, als dies bei einem regulierten Marktgeschehen der Fall wäre. Sie wollen also gerade durch

bestimmte Limitierungen der individuellen Freiheit sowohl das Gemeinwohl stärken als auch die individuelle Freiheit retten. Kommunitaristen stehen großen Unternehmen – zumal multinationalen – mehr als skeptisch gegenüber, da diese ihrer Meinung nach keine gesellschaftlichen Verpflichtungen wahrnähmen.

2.3 Anwendung und Rekonstruktion

Nach dem bisher Gesagten ist eigentlich zu erwarten, dass Vertreter quelloffener Software industrie- und unternehmensfern agierten; Ähnliches gilt für die Distanz zu Regierungen bzw. staatlichen Institutionen. Tatsächlich aber wird sehr bewusst die Nähe und Unterstützung sowohl von Unternehmen als auch von staatlichen Institutionen gesucht (Quirós und González-Barahona Upgrade 2001, S. 7; auf europäischer Ebene siehe Esteban Upgrade 2001). Zudem sind Firmen wie Suse oder Red Hat selbst inzwischen zu nicht ganz unwichtigen Mitspielern auf dem Softwareparkett avanciert. Besonders interessant ist jedoch das Engagement von IBM; allein im Jahr 2001 hat dieses Unternehmen rund eine Milliarde Euro in LINUX und Open-Source-Software investiert (Robert und Schütz Upgrade 2001, S. 16). Die Anlehnung vieler Anhänger non-proprietärer Software an IBM ist insofern bemerkenswert, als dass dieses Unternehmen selbst lange Zeit als Quasi-Monopolist verrufen war und Antitrust-Prozesse über sich hat ergehen lassen müssen – das Gedächtnis von rebellischen Bewegungen ist vielleicht prinzipiell kurz. Neben der Anlehnung an Unternehmen ist die Suche nach Kontakt zu staatlichen Institutionen zurzeit gut in der Bundesrepublik Deutschland zu beobachten. Vertreter der Open-Source-Gemeinde haben sich sehr aktiv dafür eingesetzt, dass quelloffene Software in öffentlichen Verwaltungen eingesetzt wird. Im Deutschen Bundestag wurde ein Teilerfolg errungen, denn die Server sollen in Zukunft mit LINUX betrieben werden.[11] In München konnte ein erheblicher Erfolg verbucht werden, als vom Stadtparlament entschieden wurde, die gesamte Stadtverwaltung mit LINUX auszustatten.[12]

Die Orientierung an Unternehmen wie IBM steht zumindest im Gegensatz zu den libertären Ansichten à la Stallman, da er ausdrücklich aus der gewinnorientierten und proprietären Softwareproduktion ausgestiegen ist. Für die Gemeinwohlorientierung hingegen wäre die Bereitschaft von IBM, an der Produktion eines Kollektivguts mitzuarbeiten, positiv. Dies gilt ebenso für die Unterstützung durch staatliche Institutionen; diese sind aus kommunitaristischer Sicht heraus moralisch sogar verpflichtet, Ressourcen für die Produktion von öffentlichen Gütern wie quelloffene Software bereitzustellen.

Pekka Himanen hat in seinem Buch „The Hacker Ethic and the Spirit of the Information Age" (2001), wenn auch nicht explizit, so doch sehr deutlich auf kommunitaristische Ideen zurückgegriffen. Dort wird ein wesentliches Motiv für die Arbeit an quelloffener Software darin gesehen, soziale Bestätigung durch andere zu gewinnen. Himanen weitet diese Hackerethik auch auf andere Lebens- und Berufsbereiche aus; so propagiert er bspw. in der Lehre an Schulen und Universitäten eine ge-

[11] Die Anwendercomputer werden weiterhin mit einem Microsoft-Betriebssystem ausgestattet (vgl. Krempl TP 2002). Allerdings sieht es so aus, als ob auch hier Änderungen zu erwarten sind, siehe http://www.ftd.de/tm/hs/1014399252374.html?nv=nl, Stand 08/2003.

[12] Siehe http://www.muenchen.de/aktuell/ms_linux.html, Stand 08/2003.

meinschaftliche und keine individuenzentrierte Ausbildung. Allgemeiner gesprochen finden sich in vielen Äußerungen der Unterstützer quelloffener Software stark gemeinwohlorientierte Ideen. Auch die Orientierung an einem Feindbild zur Erzeugung einer Norm und von sozialer Kohäsion, die das Produzieren von Software leiten soll, hat kommunitaristische Züge. Doch vieles bleibt widersprüchlich: Die Ablehnung von Autorität und die gleichzeitige fast schon als Vergötterung zu bezeichnende Personalisierung; die gewinnorientierten Unternehmen im Bereich quelloffener Software und die zuweilen utopisch anmutenden Ansichten zu Gemeineigentum; die Suche nach persönlicher Freiheit und eigener Regelsetzung und die teils als sektiererisch zu nennende Gruppenbildung. In ihrer Vielschichtigkeit und Heterogenität scheint die soziale Bewegung rund um quelloffene Software der frühen Umweltbewegung der 1980er Jahre vergleichbar. Sie ist nicht konsolidiert, durch Flügelkämpfe geprägt und wird möglicherweise noch eine Reihe von Transformationen und Abspaltungen erleben. Davon soll nun die Rede sein.

3. Eine mögliche Zukunft

Non-proprietäre Software erfährt zurzeit eine erhebliche Unterstützung; dies gilt sowohl für die Entwicklergemeinde als auch für Unternehmen und Institutionen. Für jene, die in der Zukunft entscheiden müssen, ob sie proprietäre oder non-proprietäre Software einsetzen, unterstützen oder entwickeln sollen, wäre es sicherlich hilfreich, einen Blick in die Zukunft werfen zu können. Unternehmen und Institutionen sind in hohem Maße abhängig von der Effektivität und Effizienz der von ihnen eingesetzten Hard- und Software; die Entscheidung für oder wider ein bestimmtes Produkt hat also weit reichende Konsequenzen. Auf den Ist-Zustand können sich Entscheider aber nicht verlassen, sie benötigen einigermaßen verlässliche Prognosen für die Zukunft. Nun sind Prognosen, die soziale Phänomene betreffen, notorisch ungenau und meist sogar unmöglich. Dies gilt ohne Zweifel auch für Aussagen über die Zukunft quelloffener Software. Ist man ihr positiv zugetan, so liegt es auf der Hand, eine blühende Entwicklung vorauszusagen und vielleicht sogar so weit zu gehen, den Tod proprietärer Software zu prognostizieren. Dies allerdings bedeutete, die Augen vor einigen Realitäten zu verschließen. Das folgende Szenario ist im Grundton zweifelsohne negativ; es muss so nicht eintreten, aber die Zukunft quelloffener Software könnte so aussehen. Jedenfalls ist es kein Naturgesetz, dass non-proprietäre Software für alle Zeiten zum Vorteil aller Menschen auf dieser Welt beitragen wird.

Non-proprietäre Software kann nicht einfach durch Unternehmen, Institutionen oder einzelne Personen vereinnahmt werden, da sie durch Lizenzen wie der *General Public License* (GPL) u.ä. abgesichert ist – auch wenn zurzeit genau dies angezweifelt wird (vgl. Spindler 2003).[13] Der derzeitige Bestand quelloffener Software wird also

[13] Durchaus interessant ist, dass laut einer Meldung der c't (Ausgabe 17/2003, S. 52) „[…] viele der vom VSI vorgebrachten Punkte für Anwender unkritisch sind, da die GPL nur relevant sei, wenn man die Software verändern und weiterverbreiten will." Hier wird mehr oder minder stillschweigend eine Trennung zwischen Anwendern und Entwicklern vollzogen, die für quelloffene Software eher bedenklich erscheint. Außerdem könnte in dieser Formulierung das Eingeständnis liegen, dass die Rechtssicherheit quelloffener Software doch nicht so hoch ist, wie oft propagiert wird.

auch in Zukunft im freien Zugriff bleiben – wobei „frei", wie Richard Stallman immer wieder zu Recht betont, nicht mit „kostenlos" übersetzt werden darf. Die Existenz non-proprietärer Software ist einem gemeinsamen Ziel geschuldet, dass nicht zuletzt oder gar in erster Linie durch die Gegnerschaft zu einem bestimmten Unternehmen oder auch einem Geschäftsmodell geprägt ist. Hinzu kommen allgemeine weltanschauliche Aspekte, wie sie oben beschrieben wurden: Gemeinwohlorientierung, bestimmte Auffassungen von Freiheit, Ablehnung von Autorität oder die Suche nach neuen Formen der Kooperation. Beide Motivkomplexe haben es ermöglicht, leistungsfähige Software zu produzieren, die nun auch von Unternehmen eingesetzt und unterstützt wird. Gerade aber hierin liegt eine große Gefahr für quelloffene Software, da dieser Erfolg beiden Motivkomplexen entgegensteht. Denn die Unterstützung kommerzieller Ziele mithilfe non-proprietärer Software scheint zunehmend in den Vordergrund zu rücken, dies zeigen viele Meldungen in Fachzeitschriften oder auch entsprechende Bücher (bspw. Fink 2002; Moody 2001; Young, Goldman Rohm 2000).

Wenn ein wesentlicher Teil der (Weiter-)Entwicklung quelloffener Software durch Unternehmen und für deren Interesse betrieben würde, so könnte dies bewirken, dass die Unterstützung der Entwicklergemeinde schwindet, da hier der Eindruck entstehen könnte, gerade das zu unterstützen, was man emotional und weltanschaulich ablehnt. Das verbindende Ziel der Unterstützer non-proprietärer Software ginge verloren, damit einhergehend auch die Motivation, entsprechende Software zu entwickeln, und letztlich auch die Entwicklerbasis, die die ungewöhnlich schnelle und effektive Produktion erst möglich gemacht hat. LINUX im Speziellen und non-proprietäre Software im Allgemeinen teilten dann das Schicksal von UNIX. Neue Entwicklungen fänden für die Plattformen der unterstützenden Unternehmen und Institutionen statt, quelloffene Software wäre nun ihrem Charakter nach … tot.

Zu der Gefahr des Todes durch Erfolg kommt die starke Personalisierung; um quelloffene Software zu unterstützen, bedarf es ganz offensichtlich Identifikationsfiguren: positive wie Linus Torvalds, negative wie Bill Gates. Gerade hierin liegt aber eine wesentliche Schwäche non-proprietärer Software, denn Identifikationsfiguren können ihre Symbolkraft verlieren – Torvalds, Raymond und Stallman werden schließlich auch älter. Es ist durchaus denkbar, dass sie die zumeist jüngeren Enthusiasten (vgl. Robles u.a. 2001, S. 25) der Open-Source- und Free-Software-Bewegung irgendwann nicht mehr ansprechen werden können. Die allgemeine demografische Entwicklung in den meisten Industrieländern – aus denen die erdrückende Mehrheit der Entwickler stammt (vgl. Robles u.a. 2001, S. 14 ff.) – tut ein Übriges: Die personelle Basis für die Entwicklung quelloffener Software bricht (gar nicht so) langsam, aber sicher weg, wenn es nicht gelingt, nicht nur junge Menschen als Entwickler zu gewinnen.

4. Fazit

Quelloffene Software stellt für die existierende Softwareindustrie technisch und ökonomisch eine Herausforderung dar (vgl. Wheeler 2003), ihre Produktionsweise

ist organisationssoziologisch mehr als interessant. Gleichzeitig aber ist die soziale Bewegung, die sich rund um quelloffene Software gebildet hat, sowohl in ihren Zielen als auch in ihren weltanschaulichen Grundlagen als sehr heterogen zu bezeichnen. Zurzeit ist völlig unklar, wie sich diese Bewegung in Zukunft entwickeln wird. Wichtig für all jene, die über den Einsatz, die Unterstützung oder auch Entwicklung quelloffener Software zu entscheiden haben, ist, dass es keine Garantie für eine positive Zukunft geben kann. Soziale Bewegungen sind keine Unternehmen – sie sind kaum berechenbar in ihrem Verhalten und in ihren Wandlungen, da ihre Existenzgrundlage in weltanschaulichen Überzeugungen liegt, die in vieler Hinsicht irrational sind oder doch zumindest der Klarheit ökonomischer Rationalität ermangeln. Die Selbstbindung von Unternehmen ist eindimensional: Gewinnerzielung. Die Selbstbindung der Bewegung rund um quelloffene Software ist multidimensional: z.B. Gemeinwohlorientierung, Freiheit, utopische Weltentwürfe, Autoritätsferne, Gegnerschaft, Gefolgschaft, Sendungsbewusstsein, Technikbegeisterung oder Neugier. All diese Motive sind in vielerlei Hinsicht interpretierbar, wandelbar, unstetig und widerspruchsvoll – und daher in ihrer Wirkung und Entwicklung kaum vorhersehbar.

Bei allen kritischen Worten aber bleibt festzuhalten: Quelloffene Software, ihre Produktionsweise und die dahinter liegenden philosophischen und weltanschaulichen Grundüberzeugungen stellen eine bedeutende *soziale* Herausforderung nicht nur für Unternehmen, sondern für ganze Informationsgesellschaften dar. Denn sie hinterfragt eine ökonomische Rationalität, die zunehmend als absolut und als einzige Variante des rationalen Handelns von Individuen und Gesellschaften dargestellt wird. Dies – ähnlich wie bei anderen sozialen Bewegungen – macht sicherlich ihre Attraktivität für viele – in erster Linie junge – Menschen aus. Hinsichtlich ihres Charakters als soziale Bewegung ist quelloffene Software gerade durch die Heterogenität der zu Grunde liegenden Motive ihrer Unterstützer so leistungsfähig; hinsichtlich der Verlässlichkeit ihrer (Weiter-)Entwicklung kann darin aber das wesentliche Manko gesehen werden. Beides – so meine Prognose – wird sich in der Zukunft nicht vereinbaren lassen.

Literatur

Baurmann, Michael (1994): *Die plötzliche Rückkehr der Wirklichkeit*, in: Geschichte und Gegenwart, Ausgabe 2 S. 103–112.

Bellah, Robert N. / Madsen, Richard / Sullivan, William / Swidler, Anne / Tipton, Steve M. (1994): *Gegen die Tyrannei des Marktes*, in: Zahlmann, Chr. (Hg.): Kommunitarismus in der Diskussion, Hamburg, S. 57–73.

Erben, Friedrun (2000): *Gemeinschaftlichkeit und Gerechtigkeit*, in: Beckmann, Kl. / Mohr, Th. / Werding, M. (Hg.): Individuum versus Kollektiv, Frankfurt am Main, S. 131–156.

Esteban, Juan Jesús Muñoz (2001): *European Initiatives Concerning the Use of Free Software in the Public Sector*, in: Upgrade – The European Online Magazine for the IT Professional, II (2001) 6, S. 36–40.

Etzioni, Amitai (1990): *The Moral Dimension*, New York 1990.

Fink, Martin (2002): *The Business and Economics of Linux and Open Source*, Upper Saddle River, New Jersey.

Forst, Rainer (1999): *Das grundlegende Recht auf Rechtfertigung. Zu einer konstruktivistischen Konzeption von Menschenrechten*, in: Brunkhorst, H. / Köhler, W. R. / Lutz-Bachmann, M. (Hg.): Recht auf Menschenrechte, Frankfurt am Main, S. 66–105.

Frohnen, Bruce (1996): *The New Communitarians and the Crisis of Modern Liberalism*, Lawrence/Kansas.

Gorr, Michael (1995): *Justice, Self-Ownership, and Natural Assets*, in: Social Philosophy and Policy, 12 (1995) 2, S. 267–291.

Grassmuck, Volker (2002): *Wissenskommunismus und Wissenskapitalismus*, in: Weber, K. / Nagenborg, M. / Spinner, H. F. (Hg.): Wissensarten, Wissensordnungen, Wissensregime. Opladen, S. 149–160.

Hertel, Guido / Niedner, Sven / Herrmann, Stefanie (ohne Jahr): *Motivation of Software Developers in Open Source Projects: An Internet-based Survey of Contributors to the Linux Kernel*,
online http://opensource.mit.edu/papers/rp-hertelniednerherrmann.pdf (07/2003).

Himanen, Pekka (2001): *The Hacker Ethic and the Spirit of the Information Age*, London.

Hobbes, Thomas (1992): *Leviathan oder Stoff, Form und Gewalt eines kirchlichen und bürgerlichen Staates*, Frankfurt am Main.

Jasay, Anthony de (1999): *Gerechtigkeit*, in: Analyse & Kritik. Zeitschrift für Sozialwissenschaften, 21 (1999) 2, S. 143–170.

Kant, Immanuel (1786, 1999): *Grundlegung zur Metaphysik der Sitten*, Hamburg: Felix Meiner 1999 (hier mit GMS bezeichnet, zitiert wird nach der 2. Auflage von 1786).

Kern, Lucian (2001): *Theorien der Verteilungsgerechtigkeit*, in: Druwe, U. / Kunz, V. / Plümper, Th. (Hg.): *Jahrbuch für Handlungs- und Entscheidungstheorie, Folge 1*, Opladen, S. 181–212.

Kersting, Wolfgang (1996): *Die politische Philosophie des Gesellschaftsvertrags*, Darmstadt.

King, J. J. (1999): *Freie Software ist eine politische Aktion*, in: Telepolis, 25.08.1999,
online http://www.telepolis.de/deutsch/special/wos/6475/1.html.

Krempl, Stefan (2002): *Der Bundestag öffnet sich für Freie Software*, in: Telepolis, 28. 02. 2002,
online http://www.heise.de/tp/deutsch/inhalt/te/11969/1.html.

Kymlicka, Will (1997): *Politische Philosophie heute*, Frankfurt am Main.

Leist, Anton (2003): *Einleitung: Ethik zwischen Hobbes und Kant*, in: Leist, A. (Hg.): Moral als Vertrag? Berlin, S. 1–36.

Locke, John (1690, 2000): *Two Treatises of Government*, Cambridge.

Mack, Eric (1995): *The Self-Ownership Proviso: A New and Improved Lockean Proviso*, in: Social Philosophy and Policy, 12 (1995) 1, S. 186–218.

Merton, Robert K. (1985): *Entwicklung und Wandel von Forschungsinteressen,* Frankfurt am Main.

Moody, Glyn (2001): *Rebel Code. The Inside Story of Linux and the Open Source Revolution*, Cambridge/Massachusetts.

Morimura, Susumu (1993): *Self-Ownership and Libertarianism*, in: Archiv für Rechts- und Sozialphilosophie, 1 (1993), S. 91–98.
Mühlbauer, Peter (2000): *Es klingt wie eine Mischung aus „liberal" und „pubertär"*, in: Telepolis, 08.11.2000,
online http://www.telepolis.de/deutsch/special/libi/4221/1.html.
Mulhall, Stephen / Swift, Adam (2002): *Liberals & Communitarians*, Malden.
Nozick, Robert (1974): *Anarchy, State, and Utopia*, New York.
Nozick, Robert (ohne Jahr): *Anarchie, Staat, Utopia*, München.
Nussbaum, Martha C. (1999): *Gerechtigkeit oder das gute Leben*, Frankfurt am Main.
Nussbaum, Martha C. (2000): *Vom Nutzen der Moraltheorie für das Leben*, Wien.
O'Reilly, Tim (1999): *Schlüsse aus der Open-Source-Software-Entwicklung*, in: Telepolis, 12.07.1999,
online http://www.telepolis.de/deutsch/special/wos/6433/1.html.
Quirós, Pedro de las Heras / González-Barahona, Jesús M. (2001): *Free Software Today*, in: Upgrade – The European Online Magazine for the IT Professional, II (2001) 6, S. 4–11.
Raymond, Eric S. (2000a): *Homesteading the Noosphere*,
online http://www.catb.org/~esr/writings/homesteading/homesteading/homesteading.ps, (08/2003).
Raymond, Eric S. (2000b): *The Magic Cauldron*,
online http://www.catb.org/~esr/writings/homesteading/magic-cauldron/magic-cauldron.ps (08/2003).
Reese-Schäfer, Walter (1994): *Was ist Kommunitarismus?* Frankfurt am Main.
Renn, Aaron M. (1998): *„Free", „Open Source", and Philosophies of Software Ownership*, http://www.urbanophile.com/arenn/hacking/fsvos.html, (07/2003).
Robert, Gilbert / Schütz, Frédéric (2001): *Should Business Adopt Free Software?* In: Upgrade – The European Online Magazine for the IT Professional, II (2001) 6, S. 12–19.
Robles, Gregorio / Scheider, Hendrik / Tretkowski, Ingo / Weber, Niels (2001): *Who Is Doing It? A research on Libre Software developers*, Institut für Informatik und Gesellschaft, TU Berlin,
online http://widi.berlios.de/paper/study.pdf, (07/2003).
Rorty, Richard (1992): *Postmodernist Bourgeois Liberalism*, in: Arneson, R. J. (Hg.): Liberalism, Vol. III. Aldershot 1992, S. 236–242.
Rössler, Beate (1994): *Gemeinschaft und Freiheit. Zum problematischen Verhältnis von Feminismus und Kommunitarismus*, in: Zahlmann, Chr. (Hg.): Kommunitarismus in der Diskussion, Hamburg, S. 74–85.
Ryan, Alan (1994): *Self-Ownership, Autonomy, and Property Rights*, in: Social Philosophy and Policy, 11 (1994) 2, S. 240–258.
Sandel, Michael J. (1995): *Die verfahrensrechtliche Republik und das ungebundene Selbst*, in: Honneth, A. (Hg.): Kommunitarismus. Frankfurt am Main ^{3}1993, S. 18–35.
Söderberg, Johan (2002): *Copyleft vs. Copyright: A Marxist Critique*, in: First Monday, 7 (2002) 3,
online http://www.firstmonday.org/issues/issue7_3/soderberg/index.html.

Spindler, Gerald (2003): *Rechtsfragen der Open Source Software*, Hg. v. Verband der Softwareindustrie Deutschlands e. V.,
online http://www.vsi.de/inhalte/aktuell/studie_final_safe.pdf (08/2003).

Spinner, Helmut F. (1994): *Die Wissensordnung*, Opladen.

Spinner, Helmut F. (2002): *Wissenskommunismus für Wissenskapitalisten – Anachronismus oder Futurismus des Informationszeitalters?* In: Dencker, K. P. (Hg.): Die Politik der Maschine, Band 5 der Interface-Reihe, Hamburg 2002, S. 295–324.

Stallman, Richard M. (1992): *Why Software Should Be Free*,
online http://www.fsf.org/philosophy/shouldbefree.html (08/2003).

Stallman, Richard M. (2001): *Free Software and Beyond*,
online http://www.mikro.org/Events/OS/ref-texte/stallman.html (08/2003).

Stallman, Richard M. (2002): *Copyright versus community in the age of computer networks*,
online http://www.carnall.demon.co.uk/stallman/all.html (08/2003).

Stallman, Richard M. (2003): *Some Confusing or Loaded Words and Phrases that are Worth Avoiding*,
online http://www.fsf.org/philosophy/words-to-avoid.html (08/2003).

Stallman, Richard M. (2001): *Harm from the Hague*, in: Upgrade – The European Online Magazine for the IT Professional, II (2001) 6, S. 20–22.

Sturma, Dieter (2000): *Universalismus und Neoaristotelismus. Amartya Sen und Martha C. Nussbaum über Ethik und soziale Gerechtigkeit*, in: Kersting, W. (Hg.): Politische Philosophie des Sozialstaats, Weilerwist, S. 257–292.

Taylor, Charles (1992): *Atomism*, in: Kymlicka, W. (Hg.): Justice in Political Philosophy, Volume II, Aldershot, S. 337–360.

Taylor, Charles (1996): *Quellen des Selbst*, Frankfurt am Main ²1996.

Walzer, Michael (1992): *The Communitarian Critique of Liberalism*, in: Kymlicka, W. (Hg.): Justice in Political Philosophy, Volume II, Aldershot, S. 392–409.

Wheeler, David A. (2003): *Why Open Source Software/Free Software (OSS/FS)? Look at the Numbers!*
online http://www.dwheeler.com/oss_fs_why.html (08/2003).

Wolff, Jonathan (1991): *Robert Nozick. Property, Justice and the Minimal State*, Stanford/California.

Young, Robert / Rohm, Wendy Goldman (2000): *Der Red Hat Coup*, Bonn.

Open Source im Kapitalismus: Gute Idee – falsches System?

LYDIA HELLER und SABINE NUSS

In den letzten fünf Jahren haben Freie und Open-Source-Software (OSS) eine rasante Karriere hingelegt. Noch 1998 schrieb die Neue Zürcher Zeitung in einem Artikel über die gerade populär werdende „Linux-Community" von „einer Art Software-Kommunismus", der im Begriff sei, die Märkte zu unterwandern (Reichert 1998). Der Gedanke des freien Nutzens, der freien Weitergabe und der offenen Kooperation mutete revolutionär an und so manch einer las darin gar die „Keimzelle" einer neuen Gesellschaftsordnung.[1] Doch zunehmend erfuhren die Rebellen ihren „marktwirtschaftlichen Ritterschlag": Apple zum Beispiel baut Teile seines neuesten Betriebssystems MacOS X auf Open-Source-Software auf,[2] IBM engagiert sich mit dem Eclipse-Projekt in der Entwicklung einer Open-Source-Entwicklungsumgebung. Und sogar die Research-Abteilung der „Deutsche-Bank-Gruppe", die langfristige Trends in Wirtschaft, Finanzwelt und Gesellschaft analysiert, macht sich inzwischen Gedanken über „Free Software – Big Business?" (DB Research zit. nach Heise 2002). Open-Source-Software, so die Prognose, werde in Zukunft in Wirtschaft und öffentlicher Verwaltung mehr und mehr zum Einsatz kommen.[3]

Der Einzug von Open Source in die Geschäftswelt geht allerdings nicht ganz reibungslos vonstatten: Zum einen bedarf es einer recht ausgeklügelten Palette eigentumssichernder Maßnahmen (Lizenzen und anderer Verträge), um mit oder auf der Basis von Open Source überhaupt Geld verdienen zu können. Zum anderen ist die Verbreitung von Open Source begleitet von einer teilweise sehr emotional und ideologisch geführten Debatte zwischen Befürwortern und Gegnern von Open Source. Beispielsweise verklagte jüngst die SCO Group den Konzern IBM auf eine Milliarde US-Dollar Schadensersatz, weil IBM im Rahmen seiner Linux-Initiative „geistiges Eigentum" von SCO gestohlen haben soll (Heise 2003b).

Im Rahmen dieser Auseinandersetzung werden Argumente vorgebracht, die durchaus repräsentativ sind für die Positionen der Gegner von Open Source. So verkündet SCO auf ihrer Webseite: „We believe that the ‚progress of science' is best

---

1 Vgl. das Oekonux-Projekt (http://www.oekonux.de).

2 MacOS X basiert auf FreeBSD.

3 Dem kostenfrei verfügbaren Betriebssystem Linux sagen die Forscher hohe Wachstumsraten voraus – auch Größen der IT-Branche wie Sun, Oracle oder SAP setzen zunehmend auf Linux. Nach Information von ZDNet stieg der Umsatz im Bereich der Linux-Server im dritten Quartal dieses Jahres um fünfzig Prozent (insgesamt 743 Millionen Dollar), während der weltweite Server-Markt lediglich zwei Prozent zulegte (ZDNet 2003). Auch als Betriebssystem auf Arbeitsplatzrechnern ist Linux auf dem Vormarsch. Schätzungen zufolge wird in fünf Jahren die Software auf 20 Prozent aller Arbeitsplatzrechner weltweit installiert sein (Heise 2003c). Schließlich erwies sich der letztjährige „Linuxtag" in Karlsruhe als die einzige deutsche IT-Messe, die in diesem Jahr mit einem Zuwachs an Ausstellern – über 40 Prozent gegenüber 2002 und mit u.a. Red Hat, Hewlett Packard und Toshiba durchaus aus dem professionellen Business – aufwarten konnte (Pro-Linux 2003).

advanced by vigorously protecting the right of authors and inventors to earn a profit from their work. [...] copyright law serves public ends by providing individuals with an incentive to pursue private ones." (McBride 2003) Auf der anderen Seite stehen die Verfechter von Open Source mit der geradezu diametral entgegengesetzten Ansicht: Erst die Offenheit von Wissen, also der Verzicht auf individuelles und exklusives Eigentum daran, würde Kooperation und damit Fortschritt ermöglichen. Im Zentrum dieses Streits stehen dabei die „Intellectual Property Rights", das heißt rechtliche Instrumente zum Schutz geistigen Eigentums, wie Urheberrechte bzw. Copyrights oder Patente. Sie sind das Mittel, mit dem festgelegt werden kann, ob ein Softwarecode allgemein zugänglich sein soll oder ob er der Öffentlichkeit vorenthalten werden soll.

In vorliegendem Aufsatz wollen wir kurz Idee und Funktionsweise von Open Source erläutern. Daran anschließend soll die in der Debatte um Open Source immer wieder vorgetragene Auffassung, dass nur private Eigentumsrechte Anreiz für Innovation und Fortschritt bieten, auf den Prüfstand gestellt werden. Dazu wollen wir die dieser Annahme zu Grunde liegende Property-Rights-Theorie erläutern und aus der Perspektive der „Kritik der politischen Ökonomie" (Marx) kritisieren. Auf dieser Basis wollen wir schließlich erklären, wie die beiden gerade genannten Positionen auf unterschiedlichem und sich widersprechendem Wege zum selben Ergebnis kommen können (so soll ja in der einen Diktion offener Code und in der anderen der geschlossene Code zu Effizienz, Innovation und Fortschritt führen) und wieso es vielleicht gar nicht so sehr die Eigentumsfrage ist, um die es hier geht oder gehen sollte.

Freie und Open-Source-Software – Die Idee hinter Open Source

Mitte der 80er Jahre entstand der Begriff „Freie Software" im Umfeld des GNU-Projektes.[4] Dessen Begründer Richard Stallman war zunehmend verärgert darüber, dass im Zuge der Kommerzialisierung von Software deren Quellcode geheim gehalten wurde. Er begann, ein freies Betriebssystem zu entwickeln und entwarf die GPL (*General Public License*), eine Lizenz, welche die private, ausschließliche Aneignung von Quellcode verhinderte. Geschützt werden sollten dagegen die Rechte, ein Programm zweckungebunden zu nutzen, zu studieren, zu kopieren, zu verändern und zu verbreiten. Der Begriff „Free Software" und gerade seine deutsche Übersetzung mit „Freie Software" führte später häufig zur Annahme, „Freie Software" sei kostenlose Software. Zwar war (und ist) der Aspekt der Kostenlosigkeit tatsächlich ein wichtiger Punkt in diesem Zusammenhang, Stallman definierte die Freiheit von Software aber eher im o.g. Sinne der Freiheit des Geistes und der Wissenschaft.[5]

Anfang der 90er Jahre, mit zunehmender Verbreitung des Internets, wurden offene, kooperative Modelle der Software-Entwicklung zunehmend populär und erlebten vor allem nach dem Erfolg des „freien" Betriebssystems Linux einen Boom.

[4] Das GNU-Projekt wurde von Richard Stallman mit dem Ziel ins Leben gerufen, eine „freie" Version des Betriebssystems UNIX zu entwickeln.

[5] Zur GNU-Philosophie und Stallmans Interpretation vgl. auch: http://www.gnu.org/philosophy/why-free.html.

Bereits 1998 diskutierten einige der „Urväter“ der Freien Software, darunter Linus Torvalds, John Ousterhout und Larry Wall die Möglichkeiten und Potenziale einer Zusammenarbeit von freien Entwicklern und kommerziellen Software-Herstellern sowie die Entwicklung geeigneter Geschäftsmodelle. Auf einem „Open-Source-Gipfeltreffen“ wurde unter Federführung von Bruce Perens und Eric S. Raymond beschlossen, anstelle des bisher üblichen Begriffs „Freie Software“ den neutraleren Begriff „Open Source“ zu benutzen.[6]

„Open Source“ sollte den gemeinsamen Nenner für all das bilden, was freie Software charakterisiert, gleichzeitig aber „pragmatische“ Ziele formulieren, die auch und gerade für die Industrie erstrebenswert wären: Argumente für offene Quellen waren nun nicht mehr in erster Linie die Aufrechterhaltung der Freiheit der Wissenschaft, sondern vor allem die größere Effizienz der Entwicklung und die bessere Qualität der Produkte. Die Verwendung von „Open“ statt „Frei“ sollte zudem signalisieren, dass man keineswegs gegen eine Kommerzialisierung von auf diesem Weg entwickelter Software sei, sondern vielmehr „offen für alles“. (O'Reilly & Associates 1999)

Die Vorzüge von Open-Source-Software

Neben der Möglichkeit, die Quellcodes von Software einzusehen, zu verändern und weiterzugeben, sind das Vorhandensein und der Zugang zu einer Netzinfrastruktur unverzichtbar für die Entwicklung von Open-Source-Software. Das Internet ist Entwicklungs- und Kommunikationsumgebung, Quelle neuer Ideen und der Rekrutierungsort neuer Mitglieder für die „Open Source Community“ zugleich. Zusammen mit Freier bzw. Open-Source-Software haben sich verschiedene Formen globaler, vernetzter Zusammenarbeit und Organisation entwickelt, die als Organisationsmodus oder Infrastrukturelement z.T. selbst wieder die Voraussetzung der Entstehung neuer Projekte, Programme, Entwicklungsmodelle etc. sind. Aus der spezifischen Herstellungsweise von Open-Source-Software und ihren Merkmalen als Produkt leiten deren Befürworter üblicherweise folgende Vorteile im Vergleich zu geschlossener bzw. proprietärer Software ab:

Hohe Sicherheit: Auf Grund des verfügbaren Codes können Programmierer oder Nutzer mit dem entsprechenden Know-how einsehen, wie das Programm funktioniert. D.h. die Kontrolle über den eigenen Computer ist gewährleistet.[7]

Flexibilität: Da der Code offen ist, kann er auf individuelle Zwecke und Bedürfnisse hin verändert werden.[8]

---

6 Vgl. Bruce Perens (1999).

7 Einer Umfrage der Evans Data Corporation unter ca. 500 Entwicklern in Nordamerika zufolge halten diese das offene Betriebssystem Linux im Vergleich zu Windows XP für weitaus sicherer. Fast ein Viertel (23%) von ihnen hielt Linux für „das sicherste System“, nur 8% hielten Windows XP für sicherer. Auch allgemein seien „Open-Source-Produkte beliebter geworden“, so die Umfrage. 2001 nutzten nur 38% der befragten Entwickler Open-Source-Software, nun sind es bereits 62% (Evans Data Corporation 2003). Zur Diskussion von Sicherheit und Open Source siehe auch den Beitrag von Gehring im vorliegenden Band.

8 „Open-source represents one of the most interesting and influential trends in the software industry over the past decade. Today, many organizations are looking toward open-source as a way to provide greater flexibility in their development practices, jump-start their development efforts by reusing existing code, and provide access to a much broader market of users“ (Brown/Booch 2002,

Hohes Entwicklungstempo bei hoher Qualität: Offener Code ist immer „work in progress". Kontinuierliche Verbesserungen, Erweiterungen und Fehlerbereinigungen erfordern dabei kontinuierliche Veröffentlichungen („release early, release often"). Open-Source-Entwickler veröffentlichen verbesserten, fehlerbereinigten Code zumeist in neuen Programmversionen, die sie eher nach Gesichtspunkten der Qualität freigeben und nicht nach Gesichtspunkten kommerzieller Verwertungszwänge (vgl. Mockus et al. 2002).

Geringe Kosten: Jeder kann den Quellcode von Open-Source-Software aus dem Netz laden. Nutzer, die nicht so vertraut mit dem Computer und dem Umgang mit seinen Anwendungsprogrammen sind, können Support, Dokumentationen und Handbücher für das entsprechende Open-Source-Programm erhalten. Sie bezahlen nur dafür, nicht aber für den Code.[9]

Schnelle und günstige Hilfe: Es gibt eine umfangreiche Gemeinde von Open-Source-Entwicklern (die „Community"), welche in Newsgroups und Mailinglisten organisiert sind und jedem, der Hilfe braucht oder Fragen hat, zu helfen versucht. Kommerzieller proprietärer Software hingegen sagt man nach, dass der Support meist von schlechter Qualität und teuer ist.[10]

Kooperation: Da die Teilnahme an einem Open-Source-Projekt in der Regel für jeden offen ist, können solche Projekte weltweit Talente anziehen, die andernfalls niemals hätten zusammengebracht werden können. Nutzer und Entwickler von Software verschmelzen in Personalunion und erhöhen damit die Feedback-Frequenzen.[11]

Die Probleme mit Open-Source-Software

Während Open-Source-Software auf Grund ihrer gerade beschriebenen Vorzüge inzwischen auch bei großen, kommerziellen Anwendern gern als kostengünstige Ressource zur Optimierung der Geschäftsabläufe oder Erweiterung der eigenen Produktpalette eingesetzt und genutzt wird,[12] ist es andererseits weniger einfach, die

S. 123).

9 DB-Research nennt in ihrer Studie explizit die „hohen Kosteneinsparpotenziale" bzw. „nachweisliche Kosten- und vermutete Stabilitäts- und Sicherheitsvorteile" als Gründe für die künftig zu erwartenden hohen Wachstumsraten speziell von Linux im Markt für Server-Software sowie für das erwartete wachsende Interesse an Officepaketen, Datenbankprogrammen und Wissensmanagement-Software auf OS-Basis (vgl. Heise 2002).

10 Vgl. z.B. Levinson, Meredith (2001): Let's Stop Wasting $78 Billion a Year, in: CIO Magazine, October 15, 2001, online http://www.cio.com/archive/101501/wasting_content.html.

11 Zur Frage des Zusammenhangs zwischen Feedback bei der Software-Entwicklung und Softwarequalität siehe z.B. (McCormack 2001): „The most striking result to emerge from the research concerned the importance of getting a low-functionality version of the product into customers' hands at the earliest opportunity. [...] Plotting the functionality against the quality of the final product demonstrated that projects in which most of the functionality was developed and tested prior to releasing a beta version performed uniformly poorly. In contrast, the projects that performed best were those in which a low-functionality version of the product was distributed to customers at an early stage." (McCormack 2001, S. 79)

12 Die „Kosteneinsparpotenziale", von denen z.B. in der Studie der DB-Research gesprochen wird (vgl. FN 5), gehen in vielen Fällen zurück auf den Einsatz von Open-Source-Software als IT-Infrastruktur von Unternehmen (z.B. freie Web-, E-Mail- und Intranet-Server). Vor allem Internet Service Provider und Hersteller von Netzwerkkomponenten (Cisco) gründeten so in der Vergangenheit

Software selbst in ein verkaufbares Produkt zu verwandeln. Wegen der breiten (und oft auch wirklich kostenlosen) Verfügbarkeit und der nahezu unbeschränkten Vervielfältigungsmöglichkeiten erfüllt Open-Source-Software eine entscheidende Bedingung für den Verkauf nicht: Sie ist nicht knapp, und ein nicht knappes Gut[13] kann, wenn überhaupt, nur gegen einen sehr geringen Preis verkauft werden. Auf Grund dessen haben sich in den letzten Jahren die unterschiedlichsten Open-Source-Lizenzen entwickelt[14], die mittlerweile häufig als „weniger restriktiv" bezeichnet werden, da man veränderten Code nicht zwangsläufig weitergeben muss (wie dies bei der GPL verpflichtend ist). Damit sind Geschäftsmodelle möglich, die sowohl mit offenem als auch mit geschlossenem Code arbeiten (Nuss 2002). Wollte die GPL ursprünglich die Freiheit von Lehre und Forschung fördern, gilt sie im geschäftlichen Kontext als zu restriktiv.

Weil der Einsatz von Software beim Nutzer zumeist eine Menge zusätzlicher Tätigkeiten nach sich zieht – Implementierung und Anpassung an die jeweilige Einsatzumgebung, Wartung, Erweiterungen, Support etc. – haben sich in den letzten Jahren eine Reihe Verwertungsmöglichkeiten für Open-Source-Software entwickelt, die vor allem auf diesen zugehörigen Dienstleistungen – auf dem „Drum-herum" – basieren: So wuchs und wächst mit dem zunehmenden Einsatz von Open-Source-Software in kleineren und mittleren Unternehmen oder in Behörden und öffentlichen Einrichtungen, in denen nur wenig Computerfachleute bzw. Spezialisten für freie Software arbeiten, vor allem der Bedarf an professionellem Support. Liebäugelt eine solche Organisation noch mit dem Einsatz oder dem Umstieg auf Open-Source-Software, so kann sie auch zunehmend auf Firmen zurückgreifen, die angepasste Einsatzkonzepte für OSS in der jeweils spezifischen Produktionsumgebung entwickeln (z.B. Linux AG). Ein zweites bekanntes und erfolgreiches Geschäftsmodell ist die Distribution. Distributoren (z.B. Red Hat, S.u.S.E) erstellen aus dem riesigen Pool freier Software, aus den Mengen an freien Systemkomponenten, Tools und Anwendungen konsistente Pakete, die auch für technisch weniger versierte Anwender einfach zu installieren und konfigurieren sind.

Darüber hinaus reichen die Experimente für die Verwertung von Open-Source-Software von ihrem Einsatz als strategisches Mittel zur Markterschließung[15] über

ihre Geschäfte erfolgreich auf offene Standards (z.B. der freien Internet-Protokolle TCP/IP). Zudem wird OSS auch gern als Basis für ein eigenes Produkt verwendet. IBM bspw. nutzt den (freien) Apache Web-Server innerhalb seiner Web-Sphere-Produktreihe, Sendmail Inc. erweitert das eigene Sendmail, das auch weiterhin frei bleibt, zu einer kompletten E-Mail-Server-Suite mit grafischer Administrationsoberfläche und anderen Tools. Vollständig und zusammen mit den eigenen Entwicklungen in ein proprietäres Produkt integrierbar ist eine genutzte freie Softwarekomponente jedoch immer nur dann, wenn die Lizenz des Open-Source-Programms dieses zulässt (Bußkamp 2003).

13 Gemeint sind hier nicht-knappe Güter im Sinne von frei verfügbaren Gütern – kostenlos zugänglich. Jene Güter, die überall kostenlos zur Verfügung stehen, sind nicht oder kaum zu verkaufen. Es ist, als stünde jemand auf der Straße und wollte eine Tüte Luft verkaufen oder als wolle jemand den Schatten eines Baumes in einem Wald verkaufen oder eben als wolle man einen Code verkaufen, der im Netz kostenlos heruntergeladen werden kann.

14 Siehe http://www.opensource.org/licenses/index.php.

15 Freie Software wird verteilt, um Kunden damit „anzufüttern", sie an bestimmte Anwendungen, Tools etc. zu gewöhnen und so an die proprietäre Produktpalette heranzuführen. Erweiterungen, Updates etc. müssen dann gekauft werden. Ähnlich können Hardwarehersteller ihre hardwarenahe

die Entwicklung von OSS gegen Lohn[16] und dem Angebot von Schulungen und Trainings für Interessierte bis hin zur Herausgabe von Handbüchern (v.a. O'Reilly Verlag) und Zeitschriften (z.B. Linux-Magazin), dem Betrieb von Online-Diensten (z.B. slashdot.org; sourceforge.net) und Merchandising (z.B. Linux-Stoff-Pinguine, T-Shirts, usw.).

Trotz vieler, mehr oder weniger erfolgreicher Open-Source-Geschäftsmodelle – die Suche nach geeigneten Methoden, florierende Geschäfte auf Open-Source-Software zu gründen, dauert an. Die Möglichkeiten, zahlende Kunden für ein „freies Gut" zu finden, Märkte dafür zu erschließen, es in eine bestehende Produktpalette zu integrieren – also schlicht damit Geld zu verdienen, werden nach wie vor diskutiert. Einig sind sich Protagonisten und Unterstützer des Open-Source-Modells, dass darin ein „immenses Marktpotenzial" (Bußkamp 2003) steckt; wie man es am besten erschließt und ausschöpft, ist weniger klar.

Die Debatte um Open-Source-Software

Open-Source-Software stößt allerdings nicht nur auf einzelne Verwertungsinteressen, sondern in großem Ausmaß auch auf theoretisches Interesse, wobei alle Disziplinen vertreten sind: Aus u.a. Ökonomie, Philosophie, Politik, Soziologie kommend und disziplinenübergreifend haben sich Forscher und andere Interessierte des Phänomens angenommen und diskutieren Bedingungen und Bedeutung der Entwicklung von Open-Source-Software, meist im Hinblick auf verallgemeinerbare Aussagen (für die ganze Volkswirtschaft, für den Menschen schlechthin, usw.). Dabei werden immer wieder grundlegende Annahmen der herrschenden Volkswirtschafts- und Betriebswirtschaftstheorie in Frage gestellt und neu diskutiert. Die drei dabei im Zentrum der Debatten stehenden wesentlichen Merkmale von Open-Source-Software sind:

- ihre Produktion durch Entwickler, die freiwillig und zu großen Teilen auch unentgeltlich daran arbeiten;
- die zumeist über das Internet stattfindende Selbstorganisation der Produktion, entweder gänzlich außerhalb von Unternehmenshierarchien oder quer durch Unternehmenshierarchien hindurch in Vernetzung mit nicht-unternehmensorganisierten Einheiten oder Einzelpersonen;
- die Freigabe qualitativ hochwertiger Produkte zur Nachahmung, Modifikation etc., also zum allgemeinen Gebrauch, die anschließend quasi ubiquitär zur Verfügung stehen.

Der Umstand, dass bei diesem Produktionsmodell eine sehr hohe Arbeitsmotivation ohne unmittelbare geldwerte Belohnung besteht, widerspricht der Anreiztheorie der Volkswirtschafts- bzw. Betriebswirtschaftslehre, und in diesem Zusam-

Software (Treiber, Systemprogramme) freigeben, um möglichst viele neue Softwareentwickler dafür zu gewinnen, für ihre Plattform Programme zu entwickeln und so eine gute Unterstützung der verkauften Hardware zu gewährleisten. (Bsp.: VA Linux Systems, Server-Systeme von IBM, Compaq oder Dell)

16 Einige Unternehmen bezahlen „ihre" Hacker inzwischen für die Entwicklung von freier bzw. Open-Source-Software – zu Marketingzwecken, aus technischem Interesse etc. Red Hat errichtete sogar ein eigenes Labor, die Red Hat Developer Laboratories. Auch IBM, Netscape, SGI und Sun bezahlen ihre Mitarbeiter für die Entwicklung freier Software.

menhang kommen schließlich auch die privaten Eigentumsrechte ins Spiel: Das klassische Dogma der herrschenden Theorie, dass nur exklusive Verwertungsrechte Einzelner an „ihren" Produkten einen Anreiz bieten würden, bzw. Voraussetzung für Innovation und Wissensvermehrung seien, trifft auf das Open-Source-Produktionsmodell offensichtlich nicht zu, weshalb dieses als „Anomalie"[17] erklärungsbedürftig scheint.

Die neuere Betriebswirtschaftslehre hält unter Rückgriff auf Erkenntnisse aus Soziologie und Sozialpsychologie gegenwärtig mit dem Ansatz der „intrinsischen Motivation" eine Erklärung bereit. So handeln „Feldstudien" zufolge Menschen nicht nur, wenn sie dazu von außen einen – zumeist monetären – Anreiz erhalten („extrinsische Motivation"), sondern auch, wenn (Frey 2001):

- eine Tätigkeit selbst Vergnügen bereitet, d.h. „ein freudiges Fluss-Erlebnis" ermöglicht, wie beim Musizieren oder Lesen eines spannenden Romans;
- es um das Einhalten von Normen um ihrer selbst willen geht, z.B. um die ethische Norm, anderen Personen nicht mutwillig zu schaden oder, in Organisationen, um die Einhaltung von Fairness oder „Teamgeist";
- ein selbst gesetztes Ziel erreicht werden soll und zwar auch, wenn dies Schwierigkeiten bereitet, z.B. beim Erstellen einer Examensarbeit.

Gerade kreative, innovative Tätigkeiten sollen weitgehend auf dieser „intrinsischen Motivation" beruhen. Die Entwickler von Open Source erklären ihre Motivation durchaus nach diesem Muster. Quelle ihrer kreativen Energie sei „die Vielzahl von sozialen und persönlichen Vorteilen, die durch die Zusammenarbeit in einer Gemeinschaft von Gleichgesinnten entstehen", z.B. die Anerkennung der anderen für besonders guten Code, der reine „Spaß am Programmieren", die „intellektuelle Herausforderung", „technische Neugier" und der „Stolz", eine universelle, günstige, fehlerfreie Alternative zum dominierenden Windows zu schaffen (vgl. dazu auch die Studien BCG/OSDN 2002; FLOSS 2002; WIDI 2001).

Die Einschränkung der „intrinsischen Motivation" liefert der Ansatz allerdings mit dem sogenannten „Verdrängungseffekt" gleich mit: Versuche mit Schulkindern zeigten, dass eine ursprünglich freiwillig und unentgeltlich ausgeübte Tätigkeit (hier die Hausaufgaben), fast nur noch gegen Geld erledigt wird, sobald eine monetäre Belohnung darauf ausgesetzt wird. Wo sich also eine Beziehung, die auf freiwilliger Kooperation und gegenseitiger Rücksichtnahme beruhte, durch Bezahlung in eine geschäftliche Beziehung verwandelt, unterhöhlt dieser extrinsische Eingriff den intrinsischen Antrieb – oder verdrängt ihn sogar ganz (Frey 2001).

Trotz Ausbreitung und zunehmender Akzeptanz von Open Source sind Geheimhaltung und private, exklusive Verwertung von Wissen in der Softwareindustrie die gängigen Strategien. Die Durchsetzung „eigener" Standards, um im Anschluss daran möglichst viele Nutzer auf die „eigenen" Programme und Anwendungen festzulegen, gehört zur Wettbewerbsstrategie vieler Unternehmen. Nicht selten gehören genau jene Unternehmen zu den ausgesprochenen Gegnern der Entwicklung von Software nach dem Open-Source-Modell: „Open-source is an intellectual property

[17] Gemeint ist eine *Anomalie* im Sinne des Wissenschaftstheoretikers Thomas Kuhn, vgl. zu Freie Software bzw. Open Source als Anomalie auch (Nuss/Heinrich 2002; O'Reilly 2002, S. 53).

destroyer“ (Microsoft Executive Jim Allchin, zit. nach OS–Summit–Report 2002, S. 86). Wenn es aber keine gesicherten Rechte geistigen Eigentums gäbe, sei zugleich die Möglichkeit der Innovation und Gewinnerzielung verhindert. Letztlich drückt sich darin nichts anderes aus als die Kurzfassung der herrschenden bürgerlichen Eigentumstheorie. Statt nun aber Open Source als empirischen Gegenbeweis noch näher auszuführen, wollen wir im Folgenden diese herrschende Theorie der Eigentumsrechte und ihren regelmäßig beschworenen Zusammenhang mit Innovationsanreizen (Anreiztheorie) erläutern.

Die Theorie der „Property Rights“

Moderne, d.h. kapitalistische Marktgesellschaften basieren maßgeblich auf der Institution des Privateigentums.[18] Theoretisches Erklärungsmodell ihrer eigentumsrechtlichen Konstruktion bildet die so genannte Property-Rights-Theorie. Sie entstammt der neoklassischen ökonomischen Analyse von Institutionen.[19] Eine der zentralen Aussagen ist, dass nur private Eigentumsrechte einen effizienten Umgang mit knappen Ressourcen gewährleisten und Anreiz zur Erhöhung der Produktivität bieten. Eine Verteilung der Eigentumsrechte auf mehrere Personen (kollektives Eigentum) oder die Beschränkung der Durchsetzbarkeit einzelner *Property Rights* hingegen führe zu unwirtschaftlichem Verhalten hinsichtlich der betroffenen Güter. Dies liegt dem Property-Rights-Ansatz zufolge daran, dass die ökonomischen Folgen der Handlungen einzelner Individuen im Falle gemeinschaftlichen Eigentums nicht in vollem Ausmaß von den Individuen getragen werden müssen, was auf lange Sicht die gemeinschaftlich genutzten Ressourcen erschöpfen wird – die sogenannte „Tragik der Allmende“ (Hardin 1968) entwickelt sich.

Die „Arbeitstheorie des Eigentums“

Die „Väter“ dieser bis hierhin nur grob skizzierten Property-Rights-Theorie, Harold Demsetz, Armen Alchian, Ronald Coase und schließlich Douglass C. North, stehen selbst wiederum in der Tradition bürgerlicher Eigentumstheorie, die auf John Locke und dessen Schrift „Über die Regierung“ (1690) zurückgeht. Entgegen dem heutigen Verständnis, nachdem die ausschließliche Aneignung von Natur durch *ein* Individuum selbstverständlich ist, dominierte zur Zeit Lockes die naturrechtliche Auffassung: Im Naturzustand, so die Vorstellung, herrscht vollkommene Freiheit und Gleichheit, Gott hat die Erde den Menschen gemeinsam gegeben. John Lockes historische Leistung war es, individuelles Eigentum *im Rahmen des Naturrechts* zu rechtfertigen, indem er sich auf das von ihm so genannte „erste Naturgesetz“ berief: Die Schöpfung und damit auch die Menschen müssen erhalten bleiben. Dazu aber muss der Mensch sich in irgendeiner Form Nahrung verschaffen. Diese Tätigkeit nun, das Pflücken einer Frucht beispielsweise, betrachtete Locke bereits als in-

[18] Das Eigentum gehört (im Kapitalismus) zu den so genannten Grundrechten. Es wird z.B. durch Artikel 14 des Grundgesetzes („Das Eigentum [...] wird gewährleistet.“) oder Zusatzartikel 1 der Europaratskonvention zum Schutz der Menschenrechte von 1950 („Jede natürliche oder juristische Person hat ein Recht auf Achtung ihres Eigentums. [...]“) gesichert.

[19] Dies ist deshalb wichtig zu erwähnen, weil damit bereits einige Grundannahmen stillschweigend vorausgesetzt sind (s. u.).

dividuelle Aneignung, wobei diese – und dies ist der Springpunkt – das Recht auf Eigentum begründet: „Was immer er also dem Zustand entrückt, den die Natur vorgesehen und in dem sie es belassen hat, hat er mit seiner Arbeit gemischt und ihm etwas Eigenes hinzugefügt. Er hat es somit zu seinem Eigentum gemacht" (Locke 1689, 1998, § 27).

Dieser rein physische Vorgang – die Vermischung von Arbeit und Natur – diente Locke als Begründung für a) das individuelle Aneignungsrecht und b) die Effizienz von Privateigentum: Arbeit = Aneignung = Privateigentum, so die Gleichung.

Mit dieser *naturrechtlichen Legitimation* von Privateigentum löste Locke einen Paradigmenwechsel in der Theoriegeschichte des Eigentums aus (Brocker 1992). Herrschte noch bis in das 17. Jahrhundert hinein in allen eigentumsrelevanten Abhandlungen (über Differenzen hinweg) Übereinstimmung darüber, dass das Privateigentum durch Konvention, d.h. von Menschen eigenmächtig eingeführt wurde und also *positiv gesetztes* Recht war, konnte es mit und seit Locke als „natürliches" Recht behandelt werden.[20] Damit einher geht gleichermaßen, dass seither das Recht auf privates Eigentum durch Arbeit begründet wird.

Bis heute ist diese „Arbeitstheorie des Eigentums" in der bürgerlichen Eigentumsauffassung weitgehend unhinterfragt.[21] Lockes Eigentumstheorie, die „weltliche Bibel" des Bürgertums (Rifkin 2000, S. 107), ist als solche in den Kanon des bürgerlichen Rechtsdenkens eingeflossen. Das Recht auf individuelle Aneignung muss heute nicht mehr diskutiert oder gar gerechtfertigt werden, schon gar nicht naturrechtlich. Die Property-Rights-Theorie nun kann als die moderne Fort- bzw. Ausführung der bürgerlichen Eigentumstheorie angesehen werden. Einer ihrer wichtigsten Vertreter ist der bereits erwähnte amerikanische Nobelpreisträger Douglass C. North. In seiner berühmt gewordenen wirtschaftshistorischen Arbeit untersuchte er die *Wirkung*[22] von Privateigentum bzw. von „gesicherten Eigentumsrechten" und kommt zu dem Schluss, dass Länder, deren Staaten gesicherte Eigentumsrechte durchsetzen konnten und können, eine effizientere Wirtschaftsleis-

[20] Das, was der Staat schließlich zu tun hatte und womit die Menschen vom Naturzustand in den Zustand der bürgerlichen Gesellschaft wechselten, war, diese natürlichen Gesetze qua Vertrag zu schützen. So war es nach Locke die wichtigste Staatsaufgabe, das Eigentum zu schützen: „Das große und *hauptsächliche Ziel,* weshalb Menschen sich zu einem Staatswesen zusammenschließen und sich unter eine Regierung stellen, ist also die Erhaltung ihres Eigentums" (Locke 1689, 1998, § 124). Damit ist Locke's Staat wesentlich ein Staat der Besitzenden und „die *vorstaatlich* angesammelten Besitztümer werden in der staatlichen Ordnung perpetuiert" (Margedant/Zimmer 1993, S. 33, Herv. d. Verf.).

[21] Auch der heutigen Eigentumsgarantie des Art. 14 Grundgesetz kommt eine „übergesetzliche" Begründung zu, so schrieb der Bundesgerichtshof Ende der 50er Jahre dem Recht auf Eigentum eine „von staatlicher Rechtssetzung unabhängige Geltung zu" (BGHZ 6, 270 ff, zit. nach Brocker 1992, S. 345). Brocker weiter: „Dem Tenor all dieser Beurteilungen schloss sich auch das 1949 geschaffene Bundesverfassungsgericht an [...] Es bezeichnete das Eigentum als ein vor- bzw. überstaatliches Recht". Ebenso ist die Auffassung, dass Arbeit das Recht auf Eigentum begründet, in die bürgerliche Rechtsprechung als unhinterfragbare Legitimation individuellen Eigentums eingeflossen, die Arbeitstheorie des Eigentums ist in der juristischen Literatur allgegenwärtig, stellenweise bezieht man sich sogar explizit auf John Locke.

[22] Der Frage, ob individuelles Eigentum überhaupt legitim ist, musste er sich auf der Grundlage der nun durchgesetzten Prämissen bürgerlichen Rechtsdenkens nicht mehr stellen.

tung generierten und generieren als Länder, die über wenig oder keine gesicherten Eigentumsrechte verfügen.

Homo oeconomicus, Knappheit und „öffentliche Güter"

Neben der bürgerlichen Eigentumsauffassung nach Locke basiert die Property-Rights-Theorie auf zwei weiteren, aus der neoklassischen ökonomischen Theorie stammenden Prämissen: Zum einen werden nach individueller Nutzenmaximierung strebende Wirtschaftssubjekte unterstellt. Zum anderen wird eine – gemessen an der Unbegrenztheit der Bedürfnisse – herrschende Knappheit an Nutzen stiftenden Gütern (Produkten, Dienstleistungen, aber auch freier Zeit) angenommen. Davon ausgehend kommt die Theorie zu dem Ergebnis, dass die alloziierende Wirkung des Marktes durch Preise – und nach der North'schen Ergänzung auch mittels gesicherter Eigentumsrechte – zu einer effizienten Produktion und optimalen Verteilung der knappen Güter führen würde.[23]

An dieser Stelle könnte man einwenden, dass aber doch gerade bei Open-Source-Software keine Knappheit vorliege, da Software ohne Qualitätsverlust und ohne größeren, zusätzlichen Kostenaufwand beliebig oft kopierbar ist. Damit träfe bereits eine der zentralen Vorannahmen der Property-Rights-Theorie zumindest für Software gar nicht zu. Nun hat die herrschende ökonomische Theorie dies durchaus berücksichtigt und hält eine Theorie *nicht knapper* Güter bereit, wozu sich auch Informationsgüter wie Software zählen lassen. Mit dem Begriff der „öffentlichen Güter" sollen spezifische Produkte, die nicht oder sehr schwer eingegrenzt werden können (z.B. öffentliches „Wissen") theoretisch fassbar gemacht werden. Öffentliche Güter weisen sich aus durch Nichtrivalität im Konsum (verringert bei zusätzlichem Nutzer die jeden anderen Nutzen zur Verfügung stehende Menge nicht) und durch Nicht-Ausschließbarkeit von der Nutzung (ein zusätzlicher Nutzer kann effektiv nicht von der Nutzung ausgeschlossen werden).

Nun sind zwar öffentliche Güter per se nicht knapp, dennoch werden auf sie die Grundprinzipien der Property-Rights-Theorie angewendet: Bei öffentlichen Gütern – so die Theorie – wäre es zwar kurzfristig optimal, sie zu Grenzkosten (kostenlos) abzugeben. Z.B. Wissen könne sich so schnell verbreiten und dem Fortschritt der Gesellschaft insgesamt dienen. Da aber die Kosten für Nachahmung niedriger ausfallen als die Kosten für Innovation, d.h. die Produktion neuen Wissens, bestehe langfristig kein hinreichender Innovationsanreiz mehr – ohne neuen Treibstoff (Wissen) aber käme der Wachstumsmotor ins Stocken.

Zur Kompensation dieses „Trade Offs" zwischen Allgemeinwohl und Effizienz werden nun spezifische Instrumente des Privateigentumsrechts (z.B. Patente) empfohlen (Liebig 2001, S. 7) Dies entspricht der Argumentation, die vorgebracht wird, wenn sich für die Wahrung und Ausweitung von privaten Rechten an geistigem Eigentum (nicht nur im Internet) ausgesprochen wird. Auch wenn die rechtlichen Schutzinstrumente für diesen Bereich unterschiedlich sind – Urheberrechts- (konti-

[23] Das eigennützige Individuum, das nutzenmaximierend seine eigene Wohlfahrt vermehrt – so eine weitere Annahme, die auf den klassischen Ökonomen Adam Smith zurückgeht – diene so automatisch auch dem allgemeinen Wohl, da auf diese Weise der Gesellschaft als Ganzes tendenziell mehr Güter zur Verfügung stehen.

nental-europäisch) und Copyright- (anglo-amerikanisch) Regelungen, Patente, Gebrauchs- und Geschmacksmustergesetze, Markenschutz usw. – sind letztlich allesamt Instrumente, welche die in irgendeiner Form ausgedrückten Ideen auf unterschiedlichste Art und für je verschiedene Zeitspannen exklusiv verwertbar machen.

Kritik der Property-Rights-Theorie

Aber wie nun kommt es zur These der Property-Rights-Theorie, dass nur die Möglichkeit, Arbeitsprodukte exklusiv verwerten zu können, dass also nur Privateigentum zu wirtschaftlicher Effizienz führe?

Kehren wir zurück zu Douglass North: Ausgehend von den oben beschriebenen Grundannahmen der Neoklassik untersuchte er wirtschaftliche Organisationsformen in der Geschichte auf ihre „Effizienz" hin, wobei er diese immer dann als gegeben ansah, wenn das nutzenmaximierende Verhalten der Subjekte zu „Ausstoßsteigerung", d.h. zu einem Wachstum an Gütern, führte.[24] Wie also haben die jeweiligen Organisationsformen von Herrschaft und Staat das nutzenmaximierende Verhalten der Wirtschaftssubjekte beeinflusst, sodass es zu einem Wachstum an Gütern kam? Zur Beantwortung dieser Frage erweitert North das neoklassische Modell, nach dem die Produktionskosten aus den Faktoren *Boden, Arbeit* und *Kapital* resultieren, um Aufwendungen, die bei der *Transaktion der Güter* entstehen, genauer: um Aufwendungen für „Abgrenzung, Schutz und Durchsetzung der Eigentumsrechte an Gütern" (North 1992, S. 33).

Diese als „Transaktionskosten"[25] bezeichneten Kosten sind dabei um so höher, je arbeitsteiliger eine Marktwirtschaft ist, da die Tauschvorgänge komplexer und anonymer werden. Allerdings kann ein Staat Transaktionskosten senken, indem er u. a. gesicherte Eigentumsrechte etabliert, denn „exklusive Eigentumsrechte, die dem Eigentümer etwas einbringen" bieten nach North einen unmittelbaren Anreiz „zur Erhöhung von Effizienz und Produktivität" (North 1988, S. 93).

Im so genannten „Prinzipal-Agent-Modell" kommt dies zum Tragen: Der Prinzipal kann bei zunehmender Arbeitsteilung in der Marktwirtschaft die Leistung seines Agenten nicht mehr direkt messen und überwachen und muss vermehrt Kontrollkosten aufwenden. Diese können gesenkt werden, indem der Prinzipal seinem

[24] „Die Ausdrücke ‚effizient' und ‚ineffizient', wie in der vorliegenden Arbeit verwendet, dienen zum Vergleich der Auswirkungen zweier Nebenbedingungen: Im einen Fall wird maximierendes Verhalten der Teilnehmer Ausstoßsteigerungen bewirken, im anderen nicht" (North 1988, S. 7, FN 2).

[25] Zum Konzept der Transaktionskosten vgl. auch Ronald Coase, der in seinem „Coase Theorem" feststellte, dass Aufwendungen beim Transfer von Rechten entscheidend für die funktionale Effizienz von Märkten seien. Nach dieser Theorie würden Märkte Rechte dann optimal arrangieren , wenn für deren Transfer keine spürbaren Aufwendungen notwendig seien. Das sei in der Praxis jedoch nicht gegeben: „In order to carry out a market transaction it is necessary to discover who it is that one wishes to deal with, to inform people that one wishes to deal and on what terms, to conduct negotiations leading up to a bargain, to draw up the contract, to undertake the inspection needed to make sure the terms of the contract are being observed, and so on. These operations are often extremely costly, sufficiently costly at any rate to prevent many transactions that would be carried out in a world in which the pricing system worked without cost." (Coase 1960, S. 15, zit. nach Meza 1998, S. 274)

Agenten Verfügungsrechte an dessen Arbeit abtritt, da ihn das zu höherer Produktion motiviere.[26]

Nun hat zwar North bei seiner Ausführung der Property-Rights-Theorie durchaus den Anspruch gehabt, eine historische Analyse zu liefern, unseres Erachtens ist allerdings gerade der *Ahistorismus* der Property-Rights-Theorie ihr Hauptmanko. So spricht North – für die ganze Geschichte des wirtschaftlichen Wandels bis hin zur Gegenwart – durchgängig von „gesicherten" oder „effizienten" Eigentumsrechten, im Gegensatz zu „nicht" oder „weniger effizienten Eigentumsrechten". Es gilt allerdings hier für das Eigentumsrecht, was der Rechtsanthropologe Uwe Wesel allgemein für den Begriff des Rechts festgehalten hat: „Wird eine Formel, die für alle Gesellschaften gleich gültig ist, für das englische Königreich und die Horde von fünfzig Mbuti, wird sie nicht auch *gleichgültig*?" (Wesel 1985, S. 66, Herv. d. Verf.).

Mit der überzeitlichen Verwendung einer Kategorie, hier jener des Privateigentums oder, was bei North das Gleiche ist, der gesicherten Eigentumsrechte, werden historische Unterschiede in den Eigentumspraxen nivelliert, wird vom jeweiligen gesellschaftlichen Wirkungskontext abstrahiert. Dies hatte schon Karl Marx den bürgerlichen Sozialphilosophen vorgeworfen, indem er feststellte, dass *alle* Produktion Aneignung von Natur und es daher eine Tautologie sei, wenn man sagt, dass das Eigentum (Aneignung) eine Bedingung der Produktion sei. „Lächerlich" aber sei es „hiervon einen Sprung auf eine *bestimmte Form des Eigentums*, z.B. das Privateigentum zu machen" (Marx 1857/58, 1953, S. 9 f., Herv. d. Verf.). Eigentumsformen könne man erst identifizieren im Rahmen einer Analyse der *jeweiligen* Produktionsstufe, auf der sich die zu untersuchende Gesellschaft befindet.

Der ahistorische Ansatz, der demnach schon bei John Locke zu finden ist,[27] wird in der modernen Property-Rights-Theorie dann sichtbar, wenn z.B. die Definition des Eigentumsrechts betrachtet wird. Nach North beinhaltet es „das Recht des Ausschlusses Dritter" (North 1988, S. 21). Nun meinte Eigentum aber beispielsweise im Mittelalter gerade nicht die Macht zur ausschließlichen Verfügung über eine Sache.[28] In dieser Epoche stand „nicht ein Abstraktum Eigentum im Mittelpunkt der Aufmerksamkeit, sondern die Fülle konkreter einzelner Rechtsstellungen, die in der Regel um die Nutzung [...] kreisen" (Hecker 1990, S. 74). Bis in das 19. Jahrhundert hinein war im größeren Teil Europas der Boden der entscheidende Produk-

[26] Vgl. „A firm's managers act as the agents for the owners or stockholders (legally referred to as the principals) of the firm. Because of this separation of ownership from control in the modern corporation, a principal-agent problem arises. […] This problem refers to the fact that while the owners of the firm want to maximize the total profits or the present value of the firm, the managers or agents want to maximize their own personal interests, such as their salaries, tenure, influence, and reputation." (Salvatore 2003, S. 647)

[27] Marx hat zwar „Eigentum" mit Aneignung gleichsetzt, im Sinne von aneignen = sich zu eigen machen: „Eine Aneignung, die sich nichts zu eigen macht, ist contradictio in subjecto" (Marx 1857/58, 1953, S. 9), Aber er setzt damit keine *spezifische* Eigentumsform, wie beispielsweise das moderne Privateigentumsrecht, damit in eins, so wie Locke dies getan hat.

[28] Der etymologische Blick auf das Wort „Eigentum" ergibt interessante Aufschlüsse. So gibt es im Mittelalter keinen einheitlichen Terminus für „Eigentum". Eine Vielzahl von Ausdrücken und Begriffen („dominium", „proprietas", „eigen") entspricht vielmehr der konkreten Ausdrucksweise jeweiliger sehr unterschiedlicher „Eigentumsverhältnisse" mit unterschiedlichen Nutzungsregelungen (Hecker 1990, S. 46).

tionsfaktor, aber: „Es gab kein Bodeneigentum *im Sinne des modernen Eigentumsbegriffs, d.h. einer zum Ausschluss Dritter berechtigenden willkürlichen Verfügungsgewalt*." (Rittstieg 1975, S. 3, Herv. d. Verf.). Auch ethnologische oder/und rechtsanthropologische Forschungen zeigen sehr illustrativ, wie unterschiedlich Aneignungsweisen sein können und wie sehr dies in Abhängigkeit steht von der Produktionsweise der betroffenen Gesellschaften (vgl. Coontz 1994, S. 53; Mauss 1990; Wesel 1982; 1985).

Eine überzeitliche Kategorie von „gesicherten Eigentumsrechten" ist analytisch nicht haltbar, vielmehr müssen Eigentumsformen – aktuell und historisch – im Kontext der jeweils herrschenden gesellschaftlichen Verhältnisse, d.h. der Produktionsverhältnisse betrachtet werden. Mit anderen Worten: Eigentumsverhältnisse drücken immer schon spezifische Produktionsverhältnisse aus und umgekehrt.

Damit ist zugleich gesagt, dass Eigentum nicht einfach eine Sache sein kann (sei sie nun materiell oder immateriell), wie das der Alltagsverstand in der Regel wahrnimmt: „Eigentum ist das, was mir gehört. Mein Haus. Mein Auto. Mein Computer." Mit einer solchen Ausdrucksweise wird gerade ausgeblendet, dass es sich bei Eigentum um ein soziales Verhältnis – spezifisch: ein Produktionsverhältnis – handelt. Dies wird auch in der Formulierung der bürgerlichen Eigentumsgesetze deutlich. Im § 903 des Bürgerlichen Gesetzbuches heißt es:

„Der Eigentümer einer Sache kann, soweit nicht das Gesetz oder Rechte Dritter entgegenstehen, mit der Sache nach Belieben verfahren und andere von jeder Einwirkung ausschließen".

Hier wird Eigentum nicht als ein Ding begriffen, sondern als Beziehung, jedoch wird dieses Verhältnis in der weiteren Auslegung als „Beziehung einer Person zu einer Sache im Sinne einer absoluten Beherrschung" (Hecker 1990, S. 17) gefasst, als „rechtliche Herrschaft einer Person über eine Sache" (Zivilonline 2002), als *Sachherrschaftsrecht*. Eigentum jedoch ist kein Ding und auch keine Herrschaft über eine Sache,[29] es ist vielmehr ein (spezifisches) Verhältnis zwischen Menschen *bezüglich* einer Sache: „Whatever technical definition of property we may prefer, we must recognize that a property right is a relation not between an owner and a thing, but between the owner and other individuals in reference to things." (M. Cohen, zit. nach Brocker 1992, S. 573)

Eigentum im Kapitalismus

Eigentum ist aber nicht nur einfach ein soziales Verhältnis zwischen Menschen, sondern eines zwischen Menschen, die sich in *verschiedenen hierarchisch gegliederten Positionen* einer Gesellschaft befinden. In kapitalistischen Gesellschaften stehen sich Kapitalist und Arbeiter in diesen machtungleichen Positionen gegenüber.

Es sind nun aber nicht „böse Kapitalisten" mit schwarzem Hut und Zigarre im Mund gemeint und auch nicht rußverschmierte Industriearbeiter im Blaumann. Eine

[29] Streng genommen kann es diese „Herrschaft über eine Sache" gar nicht geben. Herrschaft setzt voraus, dass das, was beherrscht werden soll, mit einem Willen begabt ist. *Mein* Haus, auf das ich ein Sachherrschaftsrecht habe, kann nicht weglaufen, sollte es sich entschließen, meiner Herrschaft zu entfliehen – es kann sich nicht einmal entschließen: Ein Haus hat keinen Willen.

solche Sichtweise wird mitunter Karl Marx unterstellt, dessen Schaffenswerk häufig als moralisierende Anklage (miss)verstanden wurde. Marx' Hauptwerk, „Das Kapital", ist allerdings eine (wie der Untertitel schon sagt) „Kritik der politischen Ökonomie", wobei „politische Ökonomie" zu Marx' Zeit der Terminus war für das, was in Deutschland heute als Volkswirtschaftslehre bezeichnet wird. Es handelt sich beim „Kapital" um eine wissenschaftliche Analyse und umfassende Kritik der herrschenden ökonomischen Theorie (vgl. Heinrich 2001).

Im Marx'schen Sinne sind „Kapitalist" und „Arbeiter" als Personifikationen *ökonomischer Kategorien* zu verstehen, die sowohl zur Zeit der Industrialisierung wie auch heute im Hightech-Kapitalismus analytisch greifen.[30] Nach Marx hat der Kapitalist *als Privateigentümer der Produktionsmittel* die Macht und die Kontrolle über den Einsatz dieser Mittel und als Käufer der Arbeitskraft bestimmt er zudem den Zweck ihres Einsatzes: er lässt den Arbeitnehmer produzieren zum Zwecke der Mehrwertschöpfung (in welchem Ausmaß die geschieht, ist dann allerdings historisch unterschiedlich und abhängig von sozialen Kämpfen).

Der beim Gebrauch der Arbeitskraft erzeugte Mehrwert wird schließlich im Tausch Ware gegen Geld realisiert und sich vom Kapitalisten (als Eigentümer der Produktionsmittel) angeeignet, ohne dass dieser dafür ein Äquivalent aufbringen müsste. Mit diesem Geld wird erneut Arbeitskraft gekauft, welche wiederum Mehrwert schafft. Zweck der im Privateigentum befindlichen Produktionsmittel ist es demnach, mit dem vorab eingesetzten Kapital mehr Wert („Mehrwert") zu erwirtschaften, nicht mit dem Ziel, es für Genussmittel und Luxus auszugeben (dies ist allenfalls ein Nebeneffekt), sondern mit dem Ziel, es wiederum in den Produktionsprozess zu stecken, um weiteren Mehrwert zu schöpfen, diesen erneut in die Produktion zu investieren, usw., kurz: um zu *akkumulieren*. Allerdings ist auch die Produktion nur Mittel zum Zweck der Akkumulation, und wenn es rentablere Möglichkeiten gibt, aus Geld mehr Geld zu machen, so wird beispielsweise eher in Wertpapieranlagen investiert, denn in Produktion.

Mit dieser Akkumulation von Kapital um seiner selbst Willen hat Eigentum (an Produktionsmitteln) im Kapitalismus einen anderen Zweck, als ihn vorkapitalistische oder nichtkapitalistische Gesellschaften hatten bzw. haben. Dies ist der zentrale Unterschied.

Eine notwendige Voraussetzung für den kapitalistischen Produktionsprozess ist das Vorhandensein des *eigentumslosen* Arbeiters, des „doppelt freien Arbeiters". „Doppelt frei" deshalb, weil er frei sein muss von Subsistenzmitteln (er darf keinen Zugriff auf Produktionsmittel haben, mittels derer er sich unabhängig selbst reproduzieren könnte) und weil er frei sein muss, seine Arbeitskraft zu verkaufen, d.h. darüber frei verfügen können muss.

Die zentrale rechtsphilosophische Legitimation des bürgerlichen Eigentums, nämlich dass eigene Arbeit Eigentum begründe, trifft demnach für den Kapitalismus gar nicht zu: Der Arbeiter hat in aller Regel gerade kein Eigentumsrecht auf die

[30] „Die Gestalten von Kapitalist und Grundeigentümer zeichne ich keineswegs in rosigem Licht. Aber es handelt sich hier um die Personen nur, soweit sie die Personifikation ökonomischer Kategorien sind, Träger von bestimmten Klassenverhältnissen und Interessen." (Marx 1867, 1989, S. 16)

von ihm hergestellten Arbeitsprodukte (bei Selbständigen wird dieses Verhältnis zwischen Kapital und Arbeit in eine Person hineinverlegt).[31]

Die Property-Rights-Theorie: Ahistorismus und Tautologie

Vor diesem Hintergrund lässt sich nun auch erklären, wie der ahistorische Ansatz der Property-Rights-Theorie zustande kommt. Ihre beiden grundlegenden Prämissen – der positive Zusammenhang zwischen privater Verfügungsgewalt und Motivation zur Produktion und ein offensichtlich existierendes Machtverhältnis („Prinzipal-Agent") – entstammen der unreflektierten Verallgemeinerung in Anschauung einer historisch-konkreten Wirklichkeit, die gesellschaftliche Machtverhältnisse letztlich erst erzeugt: Der Zugang zu Produktionsmitteln ist höchst ungleich verteilt und wird spezifisch genutzt, nämlich zur Kapitalakkumulation. Eine *Voraussetzung* dafür ist das Privateigentum an Produktionsmitteln und das damit verbundene Privateigentum an den hergestellten Produkten. Nur wenn der produzierte Mehrwert in der Warenzirkulationssphäre (dem Markt) auch realisiert werden kann, werden diese Produktionsmittel eingesetzt, ergo: wird produziert. Ist die Realisierung des produzierten Mehrwerts schwer oder gar unmöglich, z.B. weil die Zirkulationssphäre „verwundet" ist (wenn, wie bei Software der Fall, die Abgrenzbarkeit und Verknappung nicht oder schwer möglich ist), wird auch nicht produziert. *Nur insofern* ist es richtig, dass Eigentumsrechte eine „effiziente Produktion"[32] gewährleisten.

Die Property-Rights-Theorie abstrahiert aber von diesem historisch-konkreten Produktionsverhältnis und macht daraus ein allgemein gültiges und eben abstraktes Prinzip „bei gesicherten Eigentumsrechten effiziente Produktion."[33] Dies erweist sich im Ganzen schließlich als Tautologie, als Zirkelschluss, der lautet: Gesicherte Eigentumsrechte sind effizient, weil Effizienz (im Sinne von Kapitalverwertung) auf gesicherten Eigentumsrechten basiert (Privateigentum im Kapitalismus). Auch für

[31] Auch auf die Entstehungszeit des Kapitalismus trifft diese Theorie nicht zu. Diese Phase der „so genannten ursprünglichen Akkumulation" (Marx) ist vielmehr davon gekennzeichnet, dass gewaltsame Aneignung und nicht Arbeit Eigentum begründete. Der doppelt freie Arbeiter musste historisch erst aus den persönlichen Abhängigkeitsverhältnissen des Feudalismus herausgerissen werden und der Prozess, der dies vollbrachte, war kein idyllischer: „So [...] schuf der große Feudalherr ein ungleich größeres Proletariat durch gewaltsames Verjagen der Bauernschaft von dem Grund und Boden, worauf sie denselben feudalen Rechtstitel besaß wie er selbst, und durch Usurpation ihres Gemeindelandes [...]" (Marx 1867, 1989, S. 746).

[32] An dieser Stelle ein Wort zum Effizienzbegriff der Property-Rights-Theorie: Der Begriff der „Effizienz" drückt ein Zweck-Mittel-Verhältnis aus, das heißt, ob eine Ökonomie effizient ist, lässt sich nur feststellen, indem der Zweck dieser Ökonomie ins Verhältnis gesetzt wird zum eingesetzten Mittel. Effizienz ist dann erreicht, wenn ihr Zweck mit jenen Mitteln erreicht wurde, die dafür eingesetzt wurden. Auch hier gilt, dass es nicht „die Wirtschaft" oder „die Produktion" gibt, sondern nur historische spezifische Produktionsweisen, die nach je eigenen Zwecken organisiert sind. Eine Steigerung des Güterausstoßes ist nun aber gerade nicht Zweck beispielsweise eines kapitalistisch organisierten Unternehmens. Dieser Zweck besteht, wie wir gesehen haben, vielmehr in der beständigen und beständig gesteigerten Verwertung des Werts. Eine effiziente kapitalistische Wirtschaft muss daher nicht einfach, wie North meint, einen möglichst großen Güterausstoß ermöglichen, sondern eine hohe Kapitalverwertung.

[33] Und mitunter wird dieser Zusammenhang auch noch in die Natur des Menschen verlegt: So gehen einzelne Vertreter der Neoklassik davon aus, dass private *property rights* aus einem genetisch verankerten biologischen Instinkt zur besseren Kontrolle von immer mehr Ressourcen entstanden seien (aus: Heinsohn/Steiger 2002, S. 128 f.).

die zentralen Annahmen der Neoklassik, der Basis der Property-Rights-Theorie, gilt dies. Von den historisch-konkreten Verhältnissen des Kapitalismus wird abstrahiert und übrig bleiben Verhaltensweisen von Individuen, die zur Menschennatur erklärt werden, Verhaltensweisen, die aber erst Ergebnis einer konkreten Produktionsweise sind: Sowohl das nutzenmaximierende Individuum als auch die Knappheit an Gütern sind nicht Voraussetzungen der kapitalistischen Wirklichkeit, sondern ihr Ergebnis. Die selbstzweck-getriebene Kapitalakkumulation (aus Geld mehr Geld machen) – Funktionslogik des Kapitals – erzeugt den Schein eines dem Individuum natürlich innewohnenden Triebs zum „Immer mehr Wollen" (nutzenmaximierendes Individuum). Ähnlich erscheint die Tatsache, dass Waren nur gegen Geld getauscht werden können (also, dass nur zahlungsfähige Nachfrage bedient wird unabhängig vom tatsächlichen Bedürfnis und vom tatsächlichen Vorhandensein von Gütern) dem oberflächlichen Beobachter als *natürliche* Knappheit der Güter, dabei handelt es sich letztlich um künstlich erzeugte Knappheit.[34]

Fazit

Aus der Perspektive der Kritik der bürgerlichen ökonomischen Theorie[35] ist privates Eigentum an Produktionsmitteln und Arbeitsprodukten im Kapitalismus die Voraussetzung zur Produktion und Realisation von Mehrwert, der Zwang zur Verwertung ist wesentliches Merkmal kapitalistischer Produktionsweise. Die Verfechter gesicherter Eigentumsrechte für Software argumentieren insofern ganz im Sinne dieser Funktionslogik:[36] Soll Software als solche, als „eigenständiges, einzelnes Produkt" verkauft werden (soll also ihr Mehrwert realisiert werden), so darf sie nicht überall kostenlos zur Verfügung stehen (auch wenn das technisch möglich wäre), dann muss sie künstlich abgegrenzt, *verknappt* werden. Alle juristischen und technologischen Instrumente und Maßnahmen, die in irgendeiner Form dem Ziel dienen, den freien Zugang bzw. die kostenlose Nutzung dieser Software einzuschränken, dienen dabei als notwendiges Mittel der Verwertung.

Aber auch wenn sich Software als solche nicht so einfach in eine gut zu verkaufende Ware verwandeln lässt, wenn sich exklusive Eigentumsansprüche darauf – wie im Fall von Open-Source-Software – nicht so einfach durchsetzen lassen, fällt sie nicht automatisch aus der kapitalistischen Verwertungslogik. Die Geschäftsmodelle „um" Open Source sind mittlerweile zahlreich (und z.T. durchaus auch erfolgreich) – in denen eben nicht die Software selbst die Ware ist (und so auch weiterhin allen frei zur Verfügung stehen kann), sondern in denen „zugehörige" Dienstleistungen als Waren verkauft werden und als solche einen ausschließenden Eigentumscharakter aufweisen, da sie ja nur der zahlungsfähigen Nachfrage zugänglich sind.[37]

[34] Die künstlich erzeugte Knappheit zeigt sich bspw. immer wieder bei der Vernichtung von Lebensmitteln oder bei der Nichtausnutzung von „unrentablen" Produktionskapazitäten.

[35] Und im Rahmen der Kritik der Property-Rights-Theorie.

[36] Wenngleich die Argumentation der Apologeten der Eigentumssicherung, wie gezeigt auch auf einer oberflächlichen, ahistorischen Theorie basiert, die häufig zudem als ideologisches Beiwerk dient, wenn sie behauptet, das individuelle Streben nach „Mehr" käme zugleich dem Gemeinwohl zugute.

[37] Auch hier gilt: Verkaufbarkeit, also Mehrwertrealisierung, setzt u.a. „Knappheit" voraus, also die Verfügbarkeit des Produktes nur für jene, die es als Ware gegen Geld tauschen können.

Open-Source-Software – auch wenn sie sich als geistiges Produkt zunächst der Warenförmigkeit, der Verkaufbarkeit zu entziehen scheint – zerstört das herrschende Eigentumsregime (inkl. „intellectual property") also keineswegs. Genauso wenig bedrohlich waren (bislang) öffentliche Bibliotheken oder ganz allgemein die Weitergabe von Wissen in Forschung und Lehre. Vielmehr ist Open-Source-Software ein aktuelles Beispiel für eine Marktgesellschaften inhärente, permanente Spannung zwischen dem privaten Einschluss von Wissen und dem gesamtgesellschaftlichen Zugang dazu. Wissensprodukte, bzw. konkret im Falle von Open Source, leistungsfähige, hochwertige Software, ist das Ergebnis von Kooperation und Interaktion von Gesellschaftsmitgliedern – isolierte Individuen wären zu solchen Resultaten gar nicht oder kaum in der Lage.

Gesellschaftliche Kooperation als Entstehungsbedingung von Wissen konfligiert im Rahmen kapitalistischer Produktionsweise jedoch mit der Notwendigkeit des Einzelkapitals, Wissen exklusiv zurückzuhalten, um es verwerten zu können: Möchte auf der mikroökonomischen Ebene der einzelne Kapitalist sein in den Waren vergegenständlichtes Wissen geheim halten, um es vor Nachahmung und Mitbewerbern zu schützen, so ist eine Volkswirtschaft auf makroökonomischer Ebene am möglichst kostensparenden Zugriff auf das gesellschaftlich erzeugte Wissen angewiesen.

Dieser Widerspruch zwischen der Zirkulationsebene (in Waren vergegenständlichtes Wissen soll exklusiv sein) und der Produktionsebene (Wissen kann nur gesellschaftlich generiert werden) wird mit Hilfe rechtlicher Maßnahmen zum Schutz geistigen Eigentums, u.a. dem Patentsystem, zu kompensieren versucht. Aus dieser Logik heraus ist auch das Recht auf geistiges Eigentum, bspw. das Copyright, inhaltlich (z.B. in Form von nutzungsrechtlichen Ausnahmebestimmungen) und zeitlich limitiert.

Für die erwähnte Spannung ist es charakteristisch, dass Patent- und Copyrightsysteme seit jeher umstrittene Felder sind.[38] Bürgerliche Ökonomen können sich nicht recht entscheiden, ob bspw. Patente unnötige Schranken für den Zugang zu Wissen darstellen und Innovationen daher eher hemmen oder ob sie notwendig sind, um Innovationen zu fördern, da sie Konkurrenten davon abhalten, Forschungsergebnisse einfach nachzuahmen ohne die Vorschuss-Investitionen getätigt zu haben:

> The most important economic question about the patent system is whether on balance, with the various twists and turns that we have mentioned, it increases or reduces economic welfare. Although there are powerful economic reasons in favor of creating property rights in inventions, there are also considerable social costs and whether the benefits exceed the costs is impossible to answer with confidence on the basis of present knowledge. (Landes/Posner 2003, S. 310)

So erklärt sich, wieso einerseits die Verfechter von *intellectual property rights* nicht ganz falsch liegen, wenn sie sagen, dass private Eigentumsrechte Innovation fördern und andererseits auch die Befürworter von Open Source, bzw. von frei zugänglichem Wissen, Recht haben, wenn sie argumentieren, dass Offenheit und darauf

[38] Vgl. Gröndahl (2002).

gründende Kooperation Fortschritt und Entwicklung generieren. Der Widerspruch dieser gegensätzlichen Ansichten spiegelt nur den Widerspruch der kapitalistischen Vergesellschaftungsform wider.

Open-Source-Software steht demnach mitten in dem beschriebenen Spannungsfeld von offener Produktionsweise und Zwang zur Verwertung (das heißt, aus Geld mehr Geld machen zu müssen). Diese „Spannung" drückt sich dabei aus in den zahlreich erscheinenden Open-Source-Lizenzen, die eine Verwertungsfähigkeit von offenem Code auf unterschiedlichste Weise und für verschiedenste Bereiche ermöglichen sollen. Aber auch in den Debatten und in den (häufig auch) juristischen Auseinandersetzungen um Open Source kommt der beschriebene Widerspruch zwischen Offenheit und Einschluss (zwecks Verwertung) zum Tragen.[39]

Damit dürfte nun deutlich geworden sein, dass die eigentumssichernden Maßnahmen gar nicht so sehr das Kernproblem darstellen – sie sind vielmehr nur das Mittel, welches dazu dient, den Zweck kapitalistischer Produktion (Gewinnerzielung) zu gewährleisten. Indem nun allgemein gegen private und ausschließende Aneignung von Wissen und insbesondere gegen den privatrechtlich abgesicherten Wegschluss von Source-Code gekämpft wird, ohne dabei den Zweck des Ganzen in Frage zu stellen, bleibt das oben beschriebene Spannungsfeld bestehen und stellt einen stets und ständig umkämpften Raum dar. Solange der Zweck der herrschenden Wirtschaftsweise nicht zur Disposition steht, kann es also nur darum gehen, den offenen Zugang zu Wissen zu erhalten und zu erweitern. Dies ist aber nicht einfach eine Frage von alternativen Geschäftsmodellen, sondern von sozialen Kämpfen um ein öffentliches Gut. Derartige Kämpfe sollten sich nicht auf das Thema Software beschränken.

Literatur

The Boston Consulting Group and the Open Source Development Network (OSDN) (2002): *Boston Consulting Group/OSDN Hacker Survey*. Boston, San Diego, download http://www.osdn.com/bcg/ (21.12.2003)

Brocker, Manfred (1992): *Arbeit und Eigentum. Der Paradigmenwechsel in der neuzeitlichen Eigentumstheorie*, Darmstadt.

Brown, W. Alan/ Booch, Grady (2002): *Reusing Open-Source Software and Practices: The Impact of Open-Source on Commercial Vendors*, in: C. Gacek (Ed.): ICSR-7, LNCS 2319, Berlin/Heidelberg 2002, S. 123–136.

Bußkamp, Dirk (2003): *Open Source Tutorial*, download http://www.dbus.de/eip/inhalt.html (21.12.2003)

[39] Projekte wie das noch recht junge „UserLinux" des Open-Source-Protagonisten Bruce Perens illustrieren diesen Widerspruch: Perens arbeitet derzeit an einer neuen Linux-Distribution, die „das ökonomische Paradigma eines Linux für Unternehmen" wiederherstellen soll. Die großen Linuxhändler und Distributoren hätten die Vorzüge des freien Betriebssystems – niedrige Kosten, offene Quellen, größere Kontrolle der Nutzer über die Software, flexibler Einsatz für spezielle Bedürfnisse von spezifischen Wirtschaftsbereichen oder kleiner Firmen – „aufgeweicht", weil auch sie sich dem Shareholder Value beugten und Profit aus einer Software zögen, die ihnen nicht gehöre (vgl. Heise 2003a).

Coase, Ronald (1960): *The Problem of Social Cost*, in: Journal of Law and Economics/3, S. 1–44.

Coontz, Stephanie (1994): *Die Entstehung des Privaten. Amerikanisches Familienleben vom 17. bis zum ausgehenden 19. Jahrhundert. Theorie und Geschichte der bürgerlichen Gesellschaft*, Münster.

Evans Data Corporation (2003): *Linux More Secure Than Windows XP*, download http://www.evansdata.com/n2/pr/releases/North%20American%20Fall%202003%20Release.shtml (26.10.2003).

International Institute of Infonomics (2002): *Free/Libre and Open Source Software: Survey and Study*. Maastricht, Berlin, Paris,
online http://www.infonomics.nl/FLOSS/index.htm (21.12.2003)

Frey, Bruno (2001): *Die Grenzen ökonomischer Anreize / Was Menschen motiviert*. Neue Zürcher Zeitung, 18.05 2001.

Gröndahl, Boris (2002): *Die Tragedy of the Anti-Commons. Kapitalistische Eigentumskritik im Patentwesen*, in: Prokla Heft 126, 32/1, S. 89–101.

Hardin, Garrett (1968): *The Tragedy of the Commons*, in: Science/162, download http://www.constitution.org/cmt/tragcomm.htm (22.12.2003).

Hecker, Damian (1990): *Eigentum als Sachherrschaft. Zur Genese und Kritik eines besonderen Herrschaftsanspruchs*, Paderborn.

Heinrich, Michael (2001): *Die Wissenschaft vom Wert: Die Marxsche Kritik der politischen Ökonomie zwischen wissenschaftlicher Revolution und klassischer Tradition*, Münster.

Heinsohn, Gunnar/ Steiger, Otto (2002): *Eigentum, Zins und Geld: ungelöste Rätsel der Wirtschaftswissenschaft*, Marburg.

Heise News-Ticker (2002): *Deutsche Bank sieht starkes Wachstum bei Open-Source-Software* (Meldung vom 1511. 2002),
online http://www.heise.de/newsticker/data/pmz-15.11.02-000/
(21.12.2003).

Heise News-Ticker (2003a): *Erster Entwurf zu UserLinux* (Meldung vom 04.12.2003), download http://www.heise.de/newsticker/data/anw-04.12.03-002/ (21.12.2003).

Heise News-Ticker (2003b): *SCO verklagt IBM wegen Linux* (Meldung vom 07.03.2003),
online http://www.heise.de/newsticker/data/wst-07.03.03-001
(23.12.2003).

Heise News-Ticker (2003c): *Siemens: Linux in fünf Jahren Nummer 2 auf Desktop-PCs* (Meldung vom 18.08.2003),
online http://www.heise.de/newsticker/data/ola-18.08.03-000
(21.12.2003).

Landes, William M./ Posner, Richard A. (2003): *The Economic Structure of Intellectual Property Law*. Belknap/Harvard Univ. Press (Hg.), Cambridge (MA) und London.

Liebig, Klaus (2001): *Geistige Eigentumsrechte: Motor oder Bremse wirtschaftlicher Entwicklung? Entwicklungsländer und das TRIPS-Abkommen*, in: Deutsches Institut für Entwicklungspolitik. Berichte und Gutachten/1.

Locke, John (1689, 1998): *Zwei Abhandlungen über die Regierung*. Herausgegeben und eingeleitet von Walter Euchner, Frankfurt am Main.

Margedant, Udo/ Zimmer, Matthias (1993): *Eigentum und Freiheit. Eigentumstheorien im 17. und 18. Jahrhundert*, Idstein

Marx, Karl (1867, 1989): *Das Kapital. Kritik der politischen Ökonomie. Erster Band.* (MEW 23), Berlin.

Marx, Karl (1857/58, 1953): *Grundrisse der Kritik der Politischen Ökonomie. Rohentwurf,* Berlin.

Mauss, Marcel (1990): *Die Gabe. Form und Funktion des Austauschs in archaischen Gesellschaften*, Frankfurt am Main

McBride, Darl (2003): *Open Letter on Copyrights* (vom 4.12.2003), download http://www.sco.com/copyright/ (5.12.2003).

McCormack, Alan (2001): *Product-Development Practices That Work: How Internet Companies Build Software*, in: MIT Sloan Management Review, 42/2, S. 75–84.

Meza, David de (1998): *Coase theorem*, in: Palgrave, Band 1, S. 270–282.

Mockus, Audris/ Fielding, Roy T./ Herbleb, James D. (2002): *Two Case Studies of Open Source Software Development: Apache and Mozilla*, in: ACM Transactions on Software Engineering and Methodology, 11/3, 309–346.

North, Douglass (1992): *Institutionen institutioneller Wandel und Wirtschaftsleistung*, Tübingen.

North, Douglass (1988): *Theorie des institutionellen Wandels. Eine neue Sicht der Wirtschaftsgeschichte*, Tübingen

Nuss, Sabine (2002): *Zur Verwertung allgemeinen Wissens. Ein kapitalistisches Geschäftsmodell mit Freier Software*, in: Das Argument, Heft 248: Das Imperium des Hightech-Kapitalismus, 44/5/6.

Nuss, Sabine/ Heinrich, Michael (2002): *Freie Software und Kapitalismus*, in: Streifzüge/1.

O'Reilly & Associates (Hg.) (1999): *Open Source: kurz & gut*, Köln.

O'Reilly, Tim (2002): *A Holistic View of Open Source*, in: Open Source Summit: Public Interest & Policy Issues, October 17–18, 2002. Washington DC, George Town University.

George Town University (2002): *Open Source Summit: Public Interest & Policy Issues*, October 17–18, 2002. Washington DC,
online http://opensource.georgetown.edu (21.12.2003).

Perens, Bruce (1999): *The Open Source Definition*, in: DiBona, Chris / Ockman, Sam / Stone, Mark (Hg..): Open Sources: Voices from the Open Source Revolution, S. 171–188, Sebastopol, CA.

Pro Linux (2003): *LinuxTag 2003 übertrifft Erwartungen* (Meldung vom 13.07.2003), download http://www.pro-linux.de/news/2003/5724.html (21.12.2003).

Reichert, Bernd (1998): *Eine Art Software-Kommunismus. Gratis-Computerprogramme beleben das Geschäft.* Neue Zürcher Zeitung, 06.03.1998.

Rifkin, Jeremy (2000): *Access. Das Verschwinden des Eigentums*, Frankfurt am Main.

Rittstieg, Helmut (1975): *Eigentum als Verfassungsproblem. Zu Geschichte und Gegenwart des bürgerlichen Verfassungsstaates*, Darmstadt.

Salvatore, Dominick (2003): *Microeconomics*, New York, Oxford.

Wesel, Uwe (1982): *Die Entwicklung des Eigentums in früheren Gesellschaften*, in: Zeitschrift für vergleichende Rechtswissenschaft, 81. Jg./H. 1, S. 17–38.

Wesel, Uwe (1985): *Frühformen des Rechts in vorstaatlichen Gesellschaften: Umrisse einer Frühgeschichte des Rechts bei Sammlern und Jägern und akephalen Ackerbauern und Hirten*, Frankfurt am Main.

TU Berlin, Fachgebiet für Informatik und Gesellschaft (2001): *Who Is Doing It? A research on Libre Software developers*, Berlin, download http://widi.berlios.de/paper/study.html (21.12.2003).

ZDNet News (2003): *Server growth beats analyst forecast*. (Meldung vom 26.11.2003), download http://zdnet.com.com/2100-1103_2-5111937.html (21.12.2003).

Zivilonline (2002): *Einführung ins Privatrecht*,
online http://zivilrecht2.uibk.ac.at/online_lehre/zivilonline/allgteil/einfpriv.html (22. 12. 2003).

Vom spielerischen Ernst des Programmierens

Uli Zappe

A part we offer is our only freedom
Jon Anderson, Tales from Topographic Oceans

1. Gerechtigkeit in einer Welt beliebig rematerialisierbarer Güter

1.1. Ein Gedankenexperiment

Als die Autoren der später zum Kult gewordenen TV-Serie *Star Trek* das Handlungsgerüst für die Sendung entwarfen, unterlief ihnen ein bemerkenswerter Denkfehler. Nicht, dass das „Beamen" (also das instantane Rematerialisieren einer Person an einem anderen Ort) als solches unvorstellbar wäre – wie uns Wissenschaftsgeschichte und -theorie gelehrt haben, können wir einzig im Reich der Mathematik beweisen, dass etwas auf ewig unmöglich bleibt. Aber *wenn* es denn machbar sein sollte, Materie in Energie samt der zugehörigen Information über die Struktur der Materie hin- und sodann wieder in die nämliche Materie zurückzuverwandeln, so wäre es, entsprechende Energiereserven vorausgesetzt, offenkundig ein Leichtes, aus der einmal Information gewordenen Materie selbige in *mehrfacher* Form zurückzugewinnen: denn ein Bauplan verbraucht sich nicht, wenn ihm gemäß etwas gebaut wird. Nun soll es hier nicht um die Frage gehen, ob ein mehrfach rematerialisierter Mr. Spock mit der Vulkanierwürde zu vereinbaren wäre – oder auch nur wünschenswert für die restlichen Crew-Mitglieder der Enterprise. Entscheidend ist vielmehr der Punkt, dass das „Beamen" – konsequent durchdacht – das Ende allen materiellen Mangels gewesen wäre, da beliebige Bedarfsgüter, einmal in Information gewandelt, beliebig (im Rahmen der Energiereserven) sich hätten vervielfältigen lassen.

Warum haben die Autoren von *Star Trek* vor dieser Konsequenz zurückgeschreckt und aus dem „Beamen" nicht mehr gemacht als eine futuristische Variante der Deutschen Bahn AG? Es steht zu vermuten, dass eine solche Welt zu radikal anders als die unsere wäre, um zur Erbauung im Vorabendprogramm einer Fernsehanstalt zu taugen. Im Selbstverständnis der Moderne seit dem ausgehenden 18. Jahrhundert konstituiert nicht mehr der Staat die (bürgerliche) Gesellschaft im Namen einer höheren Ordnung; es ist der *Markt*, der den Kitt der Gesellschaft bildet, jener Ort, an dem knappe, jedenfalls nicht in beliebiger Menge verfügbare Güter auf Grund eben ihrer Knappheit, und an ihr parametrisiert, von den Mitgliedern der Gesellschaft getauscht werden: was wissen wir schon über eine Welt, in der es diese Knappheit nicht gäbe?

Es ist instruktiv für ein Verständnis dessen, was uns in der Informationsgesellschaft heraufdämmert, sich eine solche Welt beliebig rematerialisierbarer Güter aller

Art in einem Gedankenexperiment näher vor Augen zu führen. Unterstellen wir, in einer Region dieser Welt herrsche eine Hungersnot; zwar gäbe es genügend Energie, um Lebensmittel nach entsprechenden Bauplänen rematerialisieren zu können, doch stünden nicht einmal mehr Restexemplare der fehlenden Lebensmittel zur Verfügung, aus denen sich solche Baupläne gewinnen ließen, wie sie in anderen Regionen der Welt vorhanden wären. Sind Bewohner dieser anderen Regionen nun moralisch verpflichtet, den Bedürftigen Kopien dieser Baupläne zu überlassen? Handeln sie gerecht, wenn sie sie ihnen verweigern?

Ohne einer näheren philosophischen Erörterung vorzugreifen, ist es wohl nicht allzu kühn zu behaupten, dass eine weitgehend geteilte, basale moralische Intuition das Weitergeben der Baupläne jedenfalls dann als geboten ansehen würde, wenn es für deren Besitzer keinerlei Einbuße bedeutete – was bei der Weitergabe von Information, die dadurch dem ursprünglichen Besitzer ja in keiner Hinsicht verloren geht, klarerweise der Fall ist, zumindest dann, wenn eine Geheimhaltung der Information keinen *strategischen* Vorteil am Markt birgt. Ein solcher Vorteil ist aber nicht auszumachen in einer Welt, in der ein Markt knapper Güter jeglicher Art gerade nicht mehr existiert; nichts gäbe es durch eine strategisch vorteilhafte Position zu gewinnen, was nicht ohnehin zu bekommen wäre. Um den Preis eines möglicherweise schalen utilitaristischen Beigeschmacks willen: der Quotient aus dem minimalen Aufwand (einer Kopie von Information) für den Besitzer und dem maximalen Ertrag für den Nutznießer (das Überleben) ist es, der die obige moralische Intuition durch alle theoretischen Lager hindurch so schwer abweisbar macht.

Da der Aufwand zur Weitergabe der Information bei einigermaßen effizienter Datenverarbeitung, wie wir sie in einer Welt beliebig rematerialisierbarer Güter wohl voraussetzen dürfen, de facto sogar gegen null geht, bedarf es auf der Empfängerseite nicht einmal unbedingt des maximalen Nutzens, um den Quotienten aus Aufwand und Ertrag unendlich klein und das moralische Gebot zum Überlassen der Information mithin zwingend werden zu lassen; in letzter Konsequenz reicht jedweder banale Alltagsnutzen, solange er angesichts eines unendlich kleinen Aufwands im Zähler nur selbst im Nenner von endlicher Größe bleibt. Und weil auch die letzte oben gemachte Einschränkung (und somit ein letzter möglicher Ort strategischer Verwerfungen) – die in der von Mangel betroffenen Region vorausgesetzten Energiereserven – sich jedenfalls mittelfristig über stufenweise nach entsprechenden Bauplänen rematerialisierte, effiziente Kraftwerke aufheben ließe, scheint gemäß unseren grundlegenden moralischen Intuitionen zu gelten, dass in einer Welt beliebig rematerialisierbarer Güter die unbeschränkte Weitergabe von Information stets geboten ist.

1.2. Klassische Positionen

Diese grundlegenden moralischen Intuitionen lassen sich aus unterschiedlichen moraltheoretischen Perspektiven festigen. Dass die *utilitaristische* Konzeption von Gerechtigkeit, der als moralisch gut gilt, was das gesamtgesellschaftliche Wohlergehen maximieren hilft, den obigen kursorischen Überlegungen ohnehin nahe steht, wurde bereits erwähnt. Dabei verleiht der gegen null gehende Aufwand der Informationsweitergabe dem moralischen Problem für den utilitaristischen Blick ei-

nen solch schlichten Zuschnitt, dass der klassische, intersubjektive und nutzenkardinale Utilitarismus in der Tradition eines Jeremy Bentham, der sich eine objektivierbare Skalierung und Aufrechenbarkeit von Glück unter den Mitgliedern einer Gesellschaft noch zutraut, umstandslos zu demselben Resultat führt wie avanciertere, subjektive und nutzenordinale Varianten, die nur noch persönliche, ordinale Präferenzskalen der einzelnen Gesellschaftsmitglieder bezüglich der von ihnen bevorzugten Güter postulieren müssen. Denn wie auch immer einzelne utilitaristische Varianten Nutzen parametrisieren mögen, stets ist Nutzen als Nutzen zumindest größer null, und überwiegt mithin den gegen null gehenden aufzubringenden Aufwand zur Informationsweitergabe.

Die andere in der Moderne bedeutsame, der utilitaristischen komplementäre, *formale* moraltheoretische Konzeption hat jener stets vorgehalten, Nutzbringendes und damit ohnehin Erstrebtes bloß ideologisch zum Moralischen zu überhöhen, das doch gerade in Gegensatz zu Nützlichkeitserwägungen stehen kann – bei der Unverletzlichkeit der Menschenwürde eines jeden einzelnen Gesellschaftsmitgliedes etwa, die zur genuin moralischen Forderung in genau dem Moment wird, da jener Einzelne der Gesellschaft mehr zur Last fällt denn ihr nützt. Freilich stehen der Moderne, die sich ja geradezu durch das Fehlen verbindlicher traditionaler (meist sakral gestifteter) gesamtgesellschaftlicher Wertesysteme definiert, jenseits der selbstevidenten Lustgerichtetheit des Utilitarismus keine weiteren *inhaltlich* verbindlichen Wertkriterien mehr zur Verfügung. Die – eben deswegen – *formale* moraltheoretische Konzeption war daher gezwungen, auf die so genannte *transzendentale* Argumentationsfigur zurückzugreifen, die versucht, durch ein *selbstbezügliches* (und folglich von Empirie freies) Denken *logisch* zwingende Kriterien zu gewinnen. Das klassische Beispiel dafür lieferte René Descartes mit seinem berühmten Argument „Ich denke, also bin ich“: ich kann schlechterdings nicht bestreiten, dass „ich“ – zwar nicht unbedingt als physische (wie der Science-Fiction-Film *Matrix* in einer schönen Pointe zeigt), wohl aber als denkende Entität – existiere, da der Akt des Bestreitens gerade die Existenz jener bestreitenden Instanz voraussetzt, die für nonexistent erklärt werden soll – oder, in der Terminologie der Transzendentalphilosophie ausgedrückt: Die Existenz einer denkenden Instanz ist die *Bedingung der Möglichkeit* dafür, dass überhaupt etwas in einem Denkakt bestritten werden kann, und ist daher ihrerseits nicht bestreitbar, ohne sich in einen – transzendentalen – Widerspruch zu verwickeln.

Immanuel Kant hat diese transzendentale Argumentation ausgearbeitet, um sie für die Moraltheorie fruchtbar zu machen. Da menschliche Freiheit die *Bedingung der Möglichkeit* von Moral ist – wäre ich nicht frei zu entscheiden, würde ich nie vor einer moralischen (oder sonstigen) Entscheidung stehen, die moralische Frage sich mir mithin überhaupt nicht stellen –, ich als *empirisches* Wesen aber der Determination durch die Naturgesetze unterliege, bin ich als freies *Vernunft*wesen gefordert, sobald es um Moral geht. Damit ist nicht bewiesen, dass der Mensch tatsächlich in seiner Eigenschaft als Vernunftwesen frei *ist*, wohl aber, dass er dies postulieren und von der Vernunft ausgehen *muss*, sofern er sich eine moralische Frage beantworten will. Da Vernunft aber vermittels Abstraktion – dem Aufstellen allgemeiner Gesetzmäßigkeiten – operiert, folgt daraus als formales Moralkriterium

die *Verallgemeinerbarkeit*; in Kants Worten: „Handle so, dass die Maxime deines Willens jederzeit zugleich als Prinzip einer allgemeinen Gesetzgebung gelten könne" (Kant 1788, S. 54 [1. Buch, 1. Hauptstück, §7]). Das „könne" in diesem so genannten *Kategorischen* (da unbedingt, unabhängig von individuellen Werteskalen und Zielsetzungen geltenden) *Imperativ* ist dabei strikt *logisch* (eben im Sinne einer formalen Vernunftmoral) zu verstehen: das Kriterium ist, ob die Maxime sich verallgemeinern lässt, ohne sich selbst aufzuheben. Geschlossene Verträge zum eigenen Vorteil nicht einzuhalten, ist etwa eine Maxime, die, zum Gesetz verallgemeinert, die soziale Institution des Vertrages zerstören und sich damit die eigene Voraussetzung entziehen würde. Auch kann eine jedwede moralische Maxime nicht die (Moralität eben überhaupt erst ermöglichende) menschliche Freiheit infrage stellen, ohne in einen – wiederum transzendentalen – Widerspruch mit den Bedingungen ihrer Möglichkeit zu geraten, und muss daher den Menschen logisch zwingend stets als autonomes Vernunftwesen behandeln, also seine aus der Freiheit seiner Vernunft selbst gesetzten Ziele respektieren. In Kants Terminologie heißt das, den Menschen als (eigenständigen) *Zweck* anzuerkennen und nicht nur als *Mittel* zu betrachten, als welches – dem Reich der empirischen Welt zugehörig und also unter Nutzbarmachung von Naturgesetzen beherrschbar – der Mensch in strategischer Absicht für Zwecke instrumentalisiert wird, die *außerhalb* seiner selbst liegen. Kant nennt diese Implikation des kategorischen Imperativs den *praktischen Imperativ*: „Handle so, dass du die Menschheit, sowohl in deiner Person als in der Person eines jeden anderen, jederzeit zugleich als Zweck, niemals bloß als Mittel brauchest" (Kant 1785, S. 67 [Akademie-Ausgabe S. 429]).

Nun bringt die Weitergabe von Information – wie oben bereits festgestellt – ihrem Besitzer wenn, dann ausschließlich eine *strategische* Einbuße, den Verlust eines Informations*vorsprungs* am Markt. Im vorangegangenen Kontext konnte diese Situation nur *faktisch* (annäherungsweise) ausgeschlossen werden – in einer Welt beliebig rematerialisierbarer Güter, die keine Knappheit kennt, ist ein strategischer Vorteil zu nichts nütze und insofern nicht vorhanden. Mit der soeben erörterten formalen Moraltheorie hingegen lässt sich auch *moralisch* gegen eine strategische Einbuße argumentieren.

Eine solche Einbuße würde logisch eine (nun nicht mehr vollständig realisierbare) strategische Absicht voraussetzen. Zwar ist dieses Verhalten in unserer tatsächlichen Welt am Markt alltäglich; im Wettbewerb um knappe Güter sucht jeder Marktteilnehmer seinen Ertrag zu maximieren, verhält sich insofern anderen Marktteilnehmern gegenüber strategisch und gebraucht sie somit als *Mittel* zum eigenen Gewinn. *Zugleich* ist der Markt aber jenes Zusammentreffen der Bürger, das die bürgerliche Gesellschaft allererst konstituiert. Dies freilich kann nur gelingen, weil dem strategischen Handeln am Markt ein Moment innewohnt, das es transzendiert und aufs Gesellschaftliche verweist: jeder Marktteilnehmer erkennt jeden anderen als vertragswürdiges und vertragsfähiges, und mithin freies Vernunftwesen an, und insofern als *Zweck*. Das hat seinen Grund in der sozialitätsstiftenden Reziprozität der Tauschverhältnisse: jeder Nehmende ist auch ein Gebender, und findet seine eigene Absicht zum Tausch gespiegelt im Anderen; dass ich in einen Tausch einwillige, statt das von mir benötigte Gut vom Anderen schlicht einzufordern, liegt begründet

in der wie auch immer schwachen Ahnung vom Anderen als Wesen mit eigenen Zwecken, das dieser Zwecke wegen seine (knappen) Güter ebenso wenig beliebig aus der Hand geben kann, ohne andere Güter dafür zu bekommen, wie ich selbst. Um den Preis, der das Tauschverhältnis parametrisiert, wird strategisch gerungen, aber *dass* ein Tausch stattfindet, verdankt sich im Konzept der bürgerlichen Gesellschaft der freien Zusammenkunft zweier Marktteilnehmer, die auf dieser Symmetrie des Gebens und Nehmens fußt.

In einer hypothetischen Welt beliebig rematerialisierbarer Güter hingegen ist eine solche Symmetrie nicht vorhanden. Es gibt keinen Markt in unserem Sinne, da es keine knappen Güter gibt. Nur die zu rematerialisierende Information gilt es zu verteilen. Information aber ist kein marktfähiges *Konsumgut*, das seinen Besitzer beim Tausch *wechseln* würde, sondern, insofern sie unter Rezipienten und Anwendern stattdessen sich *ausbreitet*, ein *Partizipationsgut*, das der vollkommen anderen Distributionslogik der Teilhabe gehorcht; Information konstituiert Sozialität *nicht* durch Tausch am Markt, sondern gerade im *freien Fluss* der Kommunikation, wie er etwa in dem an ihrer Distributionslogik herausgebildeten akademischen Diskurs stattfindet (zum Begriff des Partizipationsguts siehe Zappe 2003, Abschnitte I und II). Die Besitzer von Informationen *geben nichts* aus der Hand, wenn sie ihre Informationen teilen; verweigern sie dies aus *strategischem* Interesse *dennoch*, so gebrauchen sie ihr Gegenüber *ausschließlich als Mittel* zur Steigerung eigener Macht, da eine sozialitätsstiftende Symmetrie des Tausches fehlt.

Damit lässt sich aus Sicht der formalen Moraltheorie festhalten, dass zwar das strategische Verhalten am Markt unserer tatsächlichen Welt insoweit moralisch erlaubt ist, als die jeweils anderen Marktteilnehmer freilich (strategisch) als Mittel, aber *zugleich* doch *auch* (in der Symmetrie des Gebens und Nehmens) als Zweck behandelt werden, dass aber in einer hypothetischen Welt beliebig rematerialisierbarer Güter die (hier ausschließlich strategisch geleitete) Verweigerung der Weitergabe von Information *moralisch verboten* wäre, da die anderen Marktteilnehmer in diesem Fall ausschließlich als Mittel, nicht aber als Zwecke gebraucht würden.

In einer wichtigen Hinsicht geht damit dieses formale über das utilitaristische Argument hinaus: Zwar gibt es in einer hypothetischen Welt beliebig rematerialisierbarer Güter grundsätzlich keinen Mangel, und strategische Marktvorteile sind nutzlos. *Falls* aber eine relevante Anzahl von Informationsbesitzern ihre Informationen warum auch immer (aus nicht-ökonomisch inspiriertem Machtwillen etwa) abzuschotten begännen und nicht mehr frei weitergeben würden, wären andere Bewohner dieser Welt plötzlich ihrerseits wieder auf strategische Vorteile angewiesen, um sie in dem Falle, dass sie einige der abgeschotteten Informationen benötigten, gegen selbige tauschen zu können: die Abschottung von Informationen, einmal begonnen, würde sich selbst stabilisieren und legitimieren. Das utilitaristische Argument, dass die Sicherung eines eigenen strategischen Vorteils unnötig und der Verlust eines strategischen Vorteils folglich keine Einbuße sei, stünde dann auf tönernen Füßen. Zwar ließe sich nach wie vor argumentieren, dass eine utilitaristisch gebotene Maximierung des *Gesamt*wohls besser ohne wechselseitige strategische Abschottung von Information zu erreichen wäre, aber zur Durchführung eines solchen Arguments müsste nun der Ertrag aus der freien Weitergabe von Information gegen einen nicht

mehr gegen null gehenden Aufwand – den Verlust an strategischem Vorsprung – tatsächlich abgewogen werden, eine nichttriviale Aufgabe, die von den einzelnen utilitaristischen Positionen höchst unterschiedlich angegangen wird und damit auch zu unterschiedlichen Resultaten führen mag. Das formale moraltheoretische Argument dagegen ist als formales von diesen empirischen Gegebenheiten unabhängig; ihm reicht, dass Information von der ihr eigentümlichen Distributionslogik her *keinen* sozialitätsstiftenden Markt konstituieren kann, an dem die anderen Marktteilnehmer, wiewohl bei strategischer Abschottung von Information als Mittel instrumentalisiert, doch immer auch als Zweck gebraucht würden. Zudem kann wechselseitige moralische *Un*vernunft, so sehr sie sich empirisch in dieser Wechselseitigkeit auch stabilisieren mag (wie etwa bei gegenseitiger Anwendung von Gewalt), gerade nicht im vernunftbasierten Sinne des kategorischen Imperativs zu einem allgemeinen Gesetz erhoben werden. Damit lässt sich formal moraltheoretisch argumentieren, dass die oben geschilderte Situation sich wechselseitig stabilisierender Abschottungen von Information, die dem Utilitarismus Kopfzerbrechen bereiten kann, ihrerseits bereits einen moralisch nicht akzeptablen Zustand darstellt.

Mit dem praktischen Imperativ, den Menschen stets auch als eigenen Zweck zu behandeln, hat es insofern ideengeschichtlich noch eine besondere Bewandtnis, als dass es diese Überlegungen sind, die Kant zu dem von ihm geprägten Begriff der *Menschenwürde* führen, die gut 150 Jahre später als unveräußerliches Recht in der bundesdeutschen Verfassung verankert wurde. Menschenwürde zu achten bedeutet, den Menschen niemals nur als Mittel zu gebrauchen. Nimmt man diesen historisch klaren Kontext ernst, so ist mit Kant festzustellen, dass die Verweigerung der Weitergabe von Information in einer Bundesrepublik Deutschland, die es geschafft hätte, Güter beliebig zu rematerialisieren, gegen die Menschenwürde verstoßen würde und somit verfassungswidrig wäre.

1.3. Positionen der Gegenwart

Seit der Ausformung der beiden hier besprochenen, für die Moderne ausschlaggebenden moraltheoretischen Ansätze sind über 200 Jahre vergangen, in denen beide Positionen verschiedenen Einwänden ausgesetzt waren, sich ihrerseits aber natürlich ebenfalls entsprechend fortentwickelten. So stehen auch die gegenwärtig bedeutsamsten Moraltheorien noch allesamt in dieser Tradition. Die drei in der aktuellen Diskussion wirkmächtigsten – die *Neue Wohlfahrtstheorie* der neoklassischen Ökonomie, die *Theory of Justice* von John Rawls, und die *Diskursethik* von Jürgen Habermas – sollen im Folgenden abschließend daraufhin befragt werden, was sie zu unserer moralischen Problemstellung in einer Welt beliebig rematerialisierbarer Güter zu sagen hätten.

Die *Neue Wohlfahrtstheorie*, der normative Teil der heute vorherrschenden neoklassischen Ökonomie, steht in direkter Linie der utilitaristischen Tradition. Die intersubjektive und nutzenkardinale Theorie Jeremy Benthams gilt freilich schon lange als nicht mehr haltbar, und in der Tat wirkt Benthams „objektiver Katalog" der verschiedenen *Pleasures* und *Pains* (Bentham 1789, Kapitel V) aus heutiger Sicht ein wenig seltsam; Benthams Bedeutung liegt eher in seiner Rolle als Gründungsvater des

Utilitarismus. Seine voraussetzungsärmeren, nur noch subjektiven und nutzenordinalen Theorienachfolger finden ihren einflussreichsten Vertreter in dem italienischen Soziologen Vilfredo Pareto, der den utilitaristischen Grundgedanken in die Form des später nach ihm benannten Paretooptimums goss (Pareto 1909, S. 617f): ein Zustand ist dann *paretooptimal*, wenn es *nicht* mehr möglich ist, den Nutzen eines Gesellschaftsmitglieds (nach dessen eigenen Maßstäben) zu mehren, ohne damit zugleich den Nutzen anderer Gesellschaftsmitglieder (nach deren je eigenen Maßstäben) zu mindern. Offenkundig bedarf diese Formulierung des Utilitarismus-Prinzips keines intersubjektiven Vergleichs von Nutzen mehr, und auch keine kardinale Nutzenmessung; einzig muss für jedes Gesellschaftsmitglied festzustellen sein, ob eine bestimmte Situation seinen Nutzen *vermehrt* oder aber *vermindert* (um wie viel auch immer). Zugleich vermeidet die Formulierung die moraltheoretische Zumutung eines kruden additiven Utilitarismus, der für das gesteigerte Wohlergehen vieler das größere Leiden weniger in Kauf nähme, freilich um den Preis, bei der weitaus strittigeren Frage passen zu müssen, *welche* von möglicherweise zahlreichen paretooptimalen Varianten der Güterverteilung zu bevorzugen sei. Es ist jedoch leicht zu sehen, dass in einer Welt beliebig rematerialisierbarer Güter jedenfalls so lange *kein* paretooptimaler Zustand gegeben ist, wie noch nicht alle Information an alle Gesellschaftsmitglieder weitergegeben wurde: denn da die Weitergabe einen gegen null gehenden Aufwand verursacht, wird so lange noch, ohne den Nutzen der Informationsbesitzer durch eine Weitergabe zu mindern, der Nutzen der Informationsempfänger gemehrt werden können. Da sich die Neue Wohlfahrtstheorie die Paretooptimalität als normatives Kriterium zu eigen macht, ist nach ihren und mithin den Maßstäben der neoklassischen Ökonomie die Weitergabe der Information geboten.

Die *Theory of Justice* von John Rawls (Rawls 1971) versucht, die formale mit der utilitaristischen Moraltheorie zusammenzudenken, und ist damit zur meistbeachteten moraltheoretischen Position der Gegenwart geworden. Um allgemein akzeptable Kriterien für Gerechtigkeit zu gewinnen, transformiert Rawls den generalisierenden Vernunftformalismus Kants mit einem Gedankenexperiment in eine zwar hypothetische, aber doch als diskursive Situation begriffene, vorgesellschaftliche *original position* („Urzustand"), in der alle zukünftigen Gesellschaftsmitglieder gemeinsam in rationaler Weise festlegen, was in der späteren Gesellschaft als gerecht zu gelten hat. Dabei sind sie durch einen *veil of ignorance* („Schleier des Nichtwissens") von der Information darüber abgeschirmt, *welche* Rolle sie in dieser zukünftigen Gesellschaft einnehmen werden. Das zwingt sie – ganz im Sinne des praktischen Imperativs von Kant – die Zwecke *aller* denkbaren Mitglieder dieser Gesellschaft bei ihren Überlegungen zu berücksichtigen, während andererseits natürlich ihre je eigene (utilitaristische) Bewertung, welche Güter von welchem Nutzen seien, ebenso einfließt. Da sich der jeweils Einzelne in diesem vormoralischen Zustand nicht für die Maximierung eines gesellschaftlichen Gesamtwohls interessiert, sondern nur die Steigerung des eigenen im Blick hat, aber gleichzeitig eben nicht weiß, an welcher Stelle der Gesellschaft er später steht, werden, so Rawls, sich die – als risikoscheu gedachten – zukünftigen Gesellschaftsmitglieder bezüglich der gerechten Verteilung der Güter schließlich auf das von ihm so genannte *difference principle* („Differenzprinzip") eini-

gen, das auch demjenigen noch zumindest minimale Vorteile garantiert, der die schlechteste Position in der zukünftigen Gesellschaft zugeteilt bekommt: „[…] the higher expectations of those better situated are just if and only if they work as part of a scheme which improves the expectations of the least advantaged members of the community" (Rawls 1971, S. 75). Das klingt dem Paretooptimum ähnlich, hat aber eine komplementäre Stoßrichtung: Das Paretooptimum als normatives Kriterium gebietet eine *Maximierung des Gesamtwohls*, die eine Grenze nur dort findet, wo einzelne Gesellschaftsmitglieder Nutzen*einbußen* erleiden. Das Differenzprinzip hingegen fordert eine *Gleichverteilung der Güter*, von der nur dort eine Ausnahme gemacht werden darf, wo auch die schwächsten Gesellschaftsmitglieder von dieser Umverteilung noch *profitieren* (falls zum Beispiel durch ein höheres Einkommen von Ärzten die medizinische Versorgung für alle gewährleistet würde). In einer Welt beliebig rematerialisierbarer Güter ist mit diesem Kriterium erst recht die Weitergabe von Information geboten, da relativ zu einem Zustand frei geteilter Information (der auf Grund des gegen null gehenden Aufwands zur Informationsweitergabe ja ebenfalls realisierbar wäre) das strategische Zurückhalten von Information eine Ungleichverteilung der Güter zugunsten der Informationsbesitzer darstellen, jedoch die Position der anderen Gesellschaftsmitglieder dabei nicht verbessern, sondern sogar *verschlechtern* würde.

Auch die *Diskursethik* von Jürgen Habermas (Habermas 1983, Kapitel 3) geht wie die *Theory of Justice* von einem Diskurs über Fragen der Gerechtigkeit aus, den sie aber nicht in einen gedankenexperimentellen Urzustand verlegt, sondern als tatsächliche gesellschaftliche Praxis, als *kommunikatives Handeln* untersucht. In einem derartigen Diskurs – so die Pointe – ist nicht Beliebiges als gerecht postulierbar. Denn damit ein Diskurs auf vernünftige Art und Weise überhaupt stattfinden kann, müssen spezifische kommunikative Handlungsnormen immer schon akzeptiert sein, etwa die Anerkennung der Diskurspartner als gleichberechtigt insofern, dass stets das bessere Argument obsiegt ohne Ansehen der Person, die es äußert. Diese Normen können *in einem Diskurs* nicht ihrerseits bestritten werden, da sie für den Akt des Bestreitens in Anspruch genommen werden müssten. Habermas greift somit die Tradition Kants auf, löst dessen transzendentale Argumentation aus ihrem vernunftimmanenten Zusammenhang und wendet sie als transzendental*pragmatische* auf die diskursive *Praxis* der Gesellschaft an; er analysiert die *Bedingungen der Möglichkeit* eines rationalen Diskurses, die kein Diskursteilnehmer *in einem Diskurs* über Gerechtigkeit in Abrede stellen kann, ohne sich dabei den argumentativen Boden unter den eigenen Füßen hinwegzuziehen und sich in einen transzendentalpragmatischen Widerspruch zu verwickeln. Die Summe dieser Bedingungen der Möglichkeit – im Wesentlichen Gleichberechtigung aller Diskursteilnehmer und Abwesenheit jeglicher Form von Zwang – bündelt und operationalisiert Habermas zu dem – handlungs*logisch* nicht bestreitbaren – *Universalisierungsgrundsatz* der Diskursethik (seinem Pendant zu Kants *kategorischem Imperativ*): „Jede gültige Norm muss der Bedingung genügen, dass die Folgen und Nebenwirkungen, die sich aus ihrer *allgemeinen* Befolgung für die Befriedigung der Interessen *jedes Einzelnen* voraussichtlich ergeben, von *allen* Betroffenen zwanglos akzeptiert werden können" (Habermas 1983, S. 103, zitiert nach S. 131). Habermas betont, dass es sich

bei dieser *operationalisierten* Bedingung – im Gegensatz zum kategorischen Imperativ – um eine *formale Argumentationsregel* für Diskurse über Gerechtigkeit handelt, und *nicht* etwa um eine *inhaltliche Norm*, die gerade nur der *Gegenstand* solcher Diskurse sein könnte: der Moraltheoretiker kann auf Grund dieser Regel ausschließlich argumentieren, *dass* zur Klärung strittiger moralischer Fragen dieser Diskurs *tatsächlich* in der realen Gesellschaft, von *allen* Betroffenen und *zwanglos*, geführt werden *muss*, aber er kann eben deswegen das Ergebnis dieses Diskurses genauso wenig vorgeben wie ein Mächtiger der Gesellschaft.

Doch stellt sich die Frage, ob damit moraltheoretisch tatsächlich schon alles gesagt ist. Zum einen gehören zu den Bedingungen der Möglichkeit eines Diskurses auch „Verhältnisse reziproker Anerkennung" (Habermas 1983, S. 98) – Verhältnisse also, die Kants praktischem Imperativ entsprechen. Insofern gelten die oben vorgetragenen Überlegungen, dass eine ausschließlich strategisch geleitete Verweigerung der Weitergabe von Information mit dem praktischen Imperativ unvereinbar ist, hier ebenso: Habermas besteht zwar auf dem tatsächlichen Diskurs der Betroffenen, aber er geht auch davon aus, dass in einem *zwanglosen* Diskurs die Diskursteilnehmer Normen, die ihnen die (für den Diskurs konstitutive) Anerkennung verweigern, schlechterdings nicht zustimmen *können*. Zum anderen weiß auch Habermas, dass ein Diskurs, in dem wirklich alle Beteiligten wirklich zwanglos zu Wort kommen, kaum je in dieser idealen Form in einer tatsächlichen Gesellschaft stattfinden wird, und schränkt daher ein, Normen seien dann gültig, wenn sie zwanglose Akzeptanz unter den Betroffenen „finden (oder finden könnten)" (Habermas 1983, S. 103) – die Formulierung „finden *könnten*" öffnet freilich theoretischen Vorüberlegungen Tür und Tor. Habermas selbst entfaltet aus der Diskursethik immerhin eine Demokratietheorie samt einem Katalog an Grundrechten (Habermas 1992, S. 155f), zu denen auch die soziale und technische Absicherung von Lebensbedingungen gehört, die die gleichberechtigte Teilnahme an Diskursen der politischen Willensbildung ermöglichen, eine funktionierende Öffentlichkeit also – das folgt letztlich aus der im Universalisierungsgrundsatz geforderten Zwanglosigkeit des moralischen Diskurses und der Beteiligung aller Betroffenen.

In einer Welt beliebig rematerialisierbarer Güter, in der potentiell *alles* Information ist, und die daher schon allein der schieren Informationsmenge wegen so genereller wie effizienter Infrastrukturen der Informationsverarbeitung bedarf, wird sich aber eine zwanglose (und das heißt auch: nicht durch strategische Informationsverarbeitung überformte) Öffentlichkeit sozial wie technisch schwer *gewährleisten* lassen, wenn die strategische Abschottung von Information als in der Sphäre der Ökonomie allgemein gangbare Norm Geltung erlangt. Die im isolierten Einzelfall möglicherweise noch klar auszumachende Differenz von disponibler instrumenteller Information (zur Rematerialisierung eines Guts) und diskursethisch nicht disponibler politisch-öffentlicher Information verliert sich allzu leicht in der Komplexität des gesellschaftlich-informationellen Ganzen, und eine einmal sozial wie technisch implementierte generelle strategische Zurichtbarkeit von Information mag sich wie ein unmerklich dünner Schleier subtilen Zwangs über die gesellschaftliche Öffentlichkeit legen. Da die Zwanglosigkeit transzendentalpragmatisch gefordert ist, muss, um

im Sinne des Universalisierungsgrundsatzes der Norm, Informationen abschotten zu dürfen, zustimmen zu *können*, ein Befürworter dieser Norm in der Lage sein zu zeigen, dass sich eine solche Beschädigung von Öffentlichkeit dauerhaft *zuverlässig* verhindern lässt – eine schwer lösbare Aufgabe. Insoweit wäre eine solche Norm auch aus demokratietheoretischer Sicht ein Problem. Zumindest für die Bewohner einer Welt beliebig rematerialisierbarer Güter, die, da hypothetisch, ja keinen tatsächlichen Diskurs für sich selbst führen können und auf die Stimme des Gedankenexperiments angewiesen sind, lässt sich daher hier – wenn auch in einem schwächeren Sinne als bei den anderen moraltheoretischen Positionen – festhalten, dass eine Zustimmung zur Abschottung von Information nicht zu bekommen sein wird.

1.4. Zusammenschau

Natürlich kann man Moraltheorie insgesamt für eine Chimäre halten; nicht erst seit Sigmund Freud und Friedrich Nietzsche gibt es dafür Argumente. Aber *wenn* man über Gerechtigkeit reden will, dann, so zeigen die obigen Überlegungen in seltener Einmütigkeit, lässt sich festhalten: *In einer Gesellschaft, deren distributive Mechanismen sich auf das Partizipationsgut Information beschränken, da alle Konsumgüter sich hieraus rematerialisieren lassen, ist die freie Verteilung von Information moralisch geboten.* Aus Kantischer Sicht verstieße eine Abschottung von Information hier zudem gegen die Menschenwürde. Die Aussage der Diskurstheorie erfolgt unter dem Vorbehalt eines tatsächlich stattfindenden Diskurses über dieses Thema, artikuliert dafür aber andererseits eben deswegen aus demokratietheoretischer Perspektive das Problem einer Gefährdung funktionierender Öffentlichkeit. Kein Wunder, dass sich die Autoren von *Star Trek* um diese Welt herumgemogelt haben.

2. Die Ausdifferenzierung des Cyberspace

Nun leben wir nicht in einer Welt beliebig rematerialisierbarer Güter – was also mögen uns die obigen Überlegungen angehen? Viel, insofern wir uns auf dem Weg in eine Informationsgesellschaft befinden und als solche ein neues gesellschaftliches Subsystem ausbilden, das in der Tat alle Merkmale einer derartigen Welt trägt: den *Cyberspace*. Die „virtuellen" Güter des Cyberspace lassen sich allesamt in Bruchteilen einer Sekunde via Internet an fast jeden beliebigen Ort der Welt „beamen", und zu einer Rematerialisierung bedarf es in aller Regel nicht mehr als eines Doppelklicks mit der Maus zum Öffnen eines Dokuments oder Starten eines Programms. Nun war das Resultat obiger Überlegungen, dass in einer solchen Welt die freie Weitergabe von Informationen nicht nur nicht moralisch verwerflich, sondern im Gegenteil *geboten* sei. Das steht in befremdlichem Kontrast zu den überbordenden Klagen von Software- und Medienindustrie, es sei das *mangelnde* moralische Bewusstsein der „Konsumenten", das sie dazu verleite, Informationen zu kopieren. Das Gegenteil ist der Fall; die moralischen Intuitionen der Nutzer treffen die Gegebenheiten offenbar weit besser als die in paternalistische Moralpredigten gehüllten Kampagnen der Industrie zur Durchsetzung ihrer Interessen.

Natürlich liegt der entscheidende Einwand nicht fern: Der Cyberspace ist ein Subsystem beliebig reproduzierbarer Güter, aber er ist *kein* autarkes System. Die in ihm agierenden Menschen sind als physische Wesen zugleich angewiesen auf die Güter des physischen Marktes, wie sie in der bürgerlichen Gesellschaft gehandelt werden. Ohne finanzielle Mittel zur Deckung ihrer Bedürfnisse auf diesem Markt können sie auch im Cyberspace nicht existieren. Um Aktion im Cyberspace sicherzustellen, so das Argument, muss der Staat daher für ein Medium sorgen, das einen Austausch zwischen Cyberspace und bürgerlicher Gesellschaft ermöglicht, und zwar in Form des so genannten „*geistigen Eigentums*". „Geistiges Eigentum" ist ein rechtliches Konstrukt, das es erlauben soll, die Sphäre der Partizipation im Cyberspace mit der des Tausches in der bürgerlichen Gesellschaft kompatibel zu machen, indem die partizipative Distributionslogik des Cyberspace der konventionellen Distributionslogik des Warentauschs subsumiert wird (siehe hierzu Zappe 2003, Abschnitt III, S. 58 f).

Insofern „geistiges Eigentum" ein Konstrukt ist, lässt es sich freilich *nicht* umstandslos mit materialem Eigentum zur Deckung bringen. Insbesondere richtet sich die genaue Ausgestaltung der rechtlichen Konstruktion nach der sie begründenden Absicht, ein gesellschaftlich erwünschtes größtmögliches Maß an Verfügbarkeit von Information zu gewährleisten. Diese Aufgabe hat es mit der Janusgesichtigkeit „geistigen Eigentums" zu tun, einerseits zwar überhaupt erst die – jedenfalls nach überkommenem ökonomischen Verständnis erforderlichen – finanziellen Anreize zur Produktion von Information zu schaffen und diese damit zu initiieren, andererseits aber die gesellschaftlich effizienteste, also weitestmögliche Verbreitung bereits existenter Information zu behindern. Wo auf der Nutzenkurve zwischen den Polen von vollkommener Geltung und Nicht-Geltung der in dem materialen Eigentumsbegriff implizierten Normen für das Konstrukt „geistiges Eigentum" das Optimum liegt, ist eine unter Ökonomen umstrittene, empirische Frage; es wird aber unter den gegebenen Voraussetzungen eines Anreize wie Verbreitung fordernden überkommenen ökonomischen Verständnisses offenkundig weder beim einen noch beim anderen Extrem zu finden sein. Das macht deutlich, dass „geistiges Eigentum" von seinem Konstruktionsplan her *nicht* mit materialem Eigentum in eins gehen kann. Diesen Punkt betont der amerikanische Rechtswissenschaftler Lawrence Lessig in *The Future of Ideas* bei seiner Auseinandersetzung mit dem Status von „geistigem Eigentum" im Zeitalter des Internets: „[…] real property doesn't map directly onto intellectual property. For as I have described, intellectual property is a balanced form of property protection. I don't have the right to fair use [Recht zur privaten Vervielfältigung, zur Zitation etc.; U.Z.] of your car; I do have the right to fair use of your book. Your right to your car is perpetual; your right to a copyright is for a limited term. The law protecting my copyright protects it in a more limited way than the law protecting my car" (Lessig 2001, S. 187).

Als Konstrukt in pragmatischer Absicht aber kann „geistiges Eigentum" keine dem materialen Eigentum vergleichbar eigenständige normative Geltung beanspruchen. Nicht nur die Optimierung seiner Nutzenfunktion und mithin seine genaue Ausprägung sind Gegenstand ökonomischer Debatten und Studien; es ist nicht einmal hinreichend empirisch abgesichert (sondern nur bislang unterstellt), *dass* das

Konstrukt des „geistigen Eigentums“ – also die Subsumtion der partizipativen Distributionslogik des Cyberspace unter die konventionelle Distributionslogik des Warentauschs – das gesteckte Ziel größtmöglicher gesellschaftlicher Verfügbarkeit von Information besser befördert als eine strikte Trennung von Cyberspace und bürgerlicher Gesellschaft. Angesichts des rasanten informationsgesellschaftlichen Wandels fällt eine solche empirische Sicherung auch schwer; allzu leicht wird die Bestandsaufnahme von neuen Realitäten überholt. Wer immer bislang für trivial hielt, dass nur finanzielle Anreize die Produktion von Information in einem gesellschaftlich relevanten Maße gewährleisten könnten, wird von dem Phänomen *GNU/Linux* eines Besseren belehrt.

Ohne gesichertes wirtschaftswissenschaftliches Fundament aber zerfällt jegliche *normative* Geltung „geistigen Eigentums“: mag auch die Produktion von Information ohne jeden Zweifel knappe Ressourcen verschlingen, so gibt es doch nirgendwo ein *moralisches* Anrecht darauf, dergleichen Aufwand *als solchen* vergolten zu bekommen. Nur unter einer von zwei Bedingungen wäre dies der Fall: im Sinne der Reziprozität steht moralisch eine Vergütung dem zu, der im Tausch von einem Gut sich trennt – das aber ist bei der Verbreitung von Information nicht der Fall; und der kann zumindest im Sinne einer utilitaristischen Moral eine Vergütung erwarten, der den gesellschaftlichen Nutzen steigert – das aber ist bislang bei der Produktion unter dem Schirm von „geistigem Eigentum“ relativ zu alternativen Varianten der Mediatisierung zwischen Cyberspace und bürgerlicher Gesellschaft ökonomisch eben nicht geklärt. Ob also, aus *moralischer* Sicht, die Distributionslogik des Cyberspace tatsächlich über das Konstrukt „geistiges Eigentum“ der der bürgerlichen Gesellschaft subsumiert werden sollte, ist eine offene Frage. Anders gesagt: *Eine moralische Legitimation für „geistiges Eigentum“ ist nicht zu bekommen.* Ohne ein moraltheoretisch abgesichertes verbindendes Medium jedoch, das beide Subsysteme auf einen vergleichbaren Nenner brächte, können normative Überlegungen (im einfachsten Falle utilitaristische Nutzenaufrechnungen) nicht beide Subsysteme integrieren. Dies wirft ein Licht darauf, wie radikal neu die Situation ist, der wir in der Informationsgesellschaft gegenübertreten: *moraltheoretisch* ist an der Analyse der Welt beliebig rematerialisierbarer Güter für ihre Anwendung auf den Cyberspace bis auf weiteres nichts zu modifizieren, weil der Cyberspace nicht Teil der bürgerlichen Gesellschaft ist, sondern in seiner klar unterschiedenen Handlungslogik sich als *eigenständiges* gesellschaftliches Subsystem *neben* ihr herausbildet, dessen Bindeglieder zu den bisherigen Subsystemen noch im empirischen Dunkel liegen.

Nur einmal fand bislang in der Geschichte der westlichen Gesellschaften ein derartiger Prozess statt. Alle vormodernen Gesellschaften – von Stammesgesellschaften bis zum Feudalismus – waren vollständig über ein umfassendes System politischer Herrschaft integriert; die Zugehörigkeit zu einer bestimmten gesellschaftlichen Schicht entschied zugleich über die Teilhabe an politischer Macht und an gesellschaftlichem Reichtum; dabei war die soziale Durchlässigkeit der Schichtungen gering. Die radikalen technologischen Umwälzungen der industriellen Revolution aber verliehen der Sphäre der ökonomischen Reproduktion eine derartige Eigendynamik, dass sie sich schließlich von der Sphäre der politischen Macht entkoppelte:

die moderne Gesellschaft differenzierte sich aus in zwei Subsysteme für die Sphären von *Tausch* und *Macht*, die *bürgerliche Gesellschaft* und den *Staat*. Keines dieser Subsysteme kann aus einem höheren Ganzen mehr Struktur schöpfen; die entpolitisierte, nur mehr mit der wirtschaftlichen Produktion befasste bürgerliche Gesellschaft konstituiert das Soziale fortan über den kapitalistisch organisierten Warentausch am freien Markt, der Staat das Politische über bürokratisierte Macht. Diese Subsysteme, die sich funktional ergänzen, bauen nicht hierarchisch aufeinander auf, sondern stehen nebeneinander und gehorchen ihrer je eigenen Handlungslogik: ein Tausch lässt sich so wenig als demokratische Abstimmung beschreiben wie eine Wahl als ökonomischer Akt (auch wenn unverbesserliche Apologeten des Marktes Letzteres bisweilen versuchen). Erfolg in der bürgerlichen Gesellschaft (Reichtum an Gütern) und im Staat (politische Machtfülle) sind nicht mehr deckungsgleich.

Es deutet nun alles darauf hin, als würde dieser Prozess der Ausdifferenzierung des gesellschaftlichen Systems auf der Basis technologischer Entwicklungen ein zweites Mal stattfinden. Diesmal getrieben von der Revolution in der Informationstechnologie tritt neben bürgerliche Gesellschaft und Staat der Cyberspace mit seiner eigenen Handlungslogik. Zwar steht der Cyberspace mit der bürgerlichen Gesellschaft in einem engen Austausch, insofern es sich (im Gegensatz zum Staat) bei beiden um Ökonomien handelt – um Systeme also, die um Produktion und Distribution kreisen –, und beide das je andere System mit Ressourcen für dessen Produktion versorgen (informationsverarbeitende hier, materielle dort), ansonsten aber sind Gegenstand und Handlungslogik klar geschieden. Als Ökonomie virtueller Güter etabliert der Cyberspace neben den Mechanismen *Tausch* und *Macht* von bürgerlicher Gesellschaft und Staat den Mechanismus der *Partizipation* (an Informationen) als vergesellschaftendes Prinzip.

In den kulturellen Wertsphären von Wissenschaft und (zumindest teilweise auch) Kunst gab es den freien Fluss der Information freilich seit eh und je; schließlich hat sich – wie oben angemerkt – die akademische Welt ja an der partizipativen Distributionslogik herausgebildet. Doch liegen Wissenschaft und Kunst als Wertsphären systemisch auf einer anderen Ebene als bürgerliche Gesellschaft und Staat; sie sind selbst keine Prozesse der Vergesellschaftung, sondern liefern diesen Prozessen kulturelle Deutungsmuster, mit deren Hilfe selbige ihre Identitäten zu stabilisieren vermögen. Anders gesagt: Auf der systemischen Ebene von bürgerlicher Gesellschaft und Staat prägen sich jene Mechanismen aus, mit denen *alle* Mitglieder einer Gesellschaft sich in Beziehung zueinander setzen: durch interessengeleiteten Tausch und politisch formierte Macht. Zur Selbstdeutung ihrer Handlungen mögen sie dabei auf von Wissenschaft und Kunst vermittelte Perspektiven zurückgreifen; aber Wissenschaft und Kunst selbst sind *keine allgemeinen* gesellschaftlichen *Praktiken*, die zu betreiben *zwingend* wäre, um Mitglied einer Gesellschaft zu sein. Wissenschaftler und Künstler regeln ihr *gesellschaftliches* Miteinander *nicht* spezifisch dadurch, dass sie Wissenschaftler und Künstler sind, sondern durch Tausch- und Machtbeziehungen wie andere Gesellschaftsmitglieder auch.

Der Cyberspace hingegen ist ein System der Vergesellschaftung par excellence: er liefert keine Deutungsmuster, sondern setzt eine ganze Weltgesellschaft in Bezie-

hung zueinander. Als neues gesellschaftliches Subsystem aber führt er als *vergesellschaftenden* Mechanismus ein, was bislang nur in den Wertsphären von Wissenschaft und Kunst wirkmächtig war, auf Grund der informationstechnologischen Umwälzungen nun jedoch erstmals zum ökonomischen Prinzip taugt: die Partizipation (siehe hierzu Zappe 2003, Abschnitt III, S. 56f). Der Reziprozität des Tausches korreliert eine Reziprozität der Partizipation an Nutzung *und* an Produktion von Information. Das schlichte Faktum der Existenz von GNU/Linux als Phänomen von wirtschaftlich relevantem Ausmaß belegt die Ankunft des neuen, dritten Subsystems in der gesellschaftlichen Realität. Eines Subsystems, über das, in seiner Eigenständigkeit, die Moraltheorie nichts anderes zu berichten vermag als über die seltsame Welt beliebig rematerialisierbarer Güter. *Wenn* also im Zusammenhang mit den informationsgesellschaftlichen Umbrüchen *moralische* Argumente Anwendung finden sollen, so lässt sich mit ihnen die Forderung nach dem Erhalt oder gar Ausbau „geistigen Eigentums" hier und heute *nicht* stützen. Im Gegenteil: *moralisch geboten* wäre nach dem, was wir wissen, die freie Weitergabe jedweder Information, zumal aus Kantischer Perspektive nur so die Menschenwürde im Cyberspace vollständige Geltung erlangte und die für Demokratien konstitutive Öffentlichkeit möglicherweise nur so im Cyberspace frei von Gefährdung bliebe. Wer auch immer dem „geistigen Eigentum" das Wort redet, sollte sich moralischer Argumente folglich besser enthalten.

Das ist nicht unbedingt leicht zu erfassen, wie alles Neue. Gleichwohl wäre der Debatte über „geistiges Eigentum", Patente und Urheberrecht im Zeitalter des Internets etwas mehr Klarsicht und etwas weniger Selbstgerechtigkeit zu wünschen. So heißt es in einem jüngst zu diesem Thema veröffentlichten Artikel im Nachrichtenmagazin *Der Spiegel*, einer Publikation, die sich gemeinhin zumindest eine gewisse kritische Distanz zu bewahren vermag, aus offenbar vollkommen bruchlos von Software- und Medienindustrie übernommener Perspektive: „Vor allem jener Generation, die mit dem Internet bereits aufgewachsen ist, fehlt jedes Verständnis dafür, dass auch geistiges Eigentum nun mal Eigentum ist – und Datenklau letztlich nur eines: Diebstahl" (Balzli u.a. Der Spiegel 2003, S. 74). Der erste Teil dieses Satzes ist, wie nun ausführlich diskutiert, schlicht falsch, der zweite tautologisch. Als sei das nicht genug, ergeht sich der Text in den ausgeleierten konservativen Klagen über den Ausverkauf der Kultur angesichts neuer Entwicklungen: „Wird die noch von Benjamin beschriebene Autorität der Einzigartigkeit ersetzt durch Ex-und-hopp-Wegwerfkunst? Eine egalitäre Konsumkultur für den Augenblick statt Wahrheit für eine wie auch immer geartete Ewigkeit?" (ebd.). Ausgerechnet durch dasjenige gesellschaftliche Subsystem, das die Partizipation aller an der Produktion von Information geschichtlich erstmals erlaubt, dräut laut den Autoren „am Ende der Entwicklung […] die Diktatur des massenkompatiblen Pop-Proletariats eines Dieter Bohlen" (ebd., S. 75). Einen kaum noch zu unterbietenden intellektuellen Tiefpunkt erreicht der Artikel freilich, wenn er sich fragt: „Doch was ist eine Kunst noch wert, die es überall gratis gibt?" (ebd., S. 71) – „Und welche Werte hat eine Gratisgesellschaft eigentlich noch, da doch bereits der Wert ihrer Produkte gegen null tendiert?" (ebd., S. 74). Drastischer lässt sich schwerlich vorführen, dass kapitalistische Gesellschaften die unselige Tendenz haben, alle Werte – neue wie altehrwürdi-

ge – unter den Moloch des Tauschwerts zu subsumieren. Dass sich die geschilderten Vorgänge zudem in dem neu ausdifferenzierten Subsystem der Ökonomie der Partizipation und eben nicht in dem traditionellen Subsystem der Ökonomie des Tauschs abspielen, blieb den Autoren offenkundig vollständig verborgen; einer politischen Partei hingegen, die, im Subsystem der Macht agierend, potentiellen Wählern unentgeltlich ihre Konzepte darlegt, hätten die Autoren vermutlich nicht attestiert, keine Werte zu vertreten. Das zeigt, wie schwer es offenbar ist, die an zwei Subsystemen erlernte Fähigkeit des Differenzierens auf ein neu hinzugekommenes drittes Subsystem zu übertragen.

Kein Wunder, dass die Industrie angesichts solcher Verwirrung die Chance ergreift, ihre Interessen offensiv zu vertreten. So heißt es bei dem Softwarehersteller *Adobe* zur Erläuterung von Kopierschutzmaßnahmen kurz und bündig: „Wir hoffen, dass die Kunden die Notwendigkeit erkennen werden, das unberechtigte Kopieren geistigen Eigentums zu unterbinden" (Adobe 2003). Dass kein Recht auf Kopieren „geistigen Eigentums" besteht, wird hier wie selbstverständlich vorausgesetzt. (Der Klarheit halber sei angemerkt, dass eine Konsumgut-analoge Veräußerung zuvor unentgeltlich kopierter Software durch partizipationsgesellschaftliche Distributionsprinzipien logischerweise *nicht* gedeckt und also *nicht* legitimiert ist.) Die *Business Software Alliance (BSA)*, ein Zusammenschluss etlicher großer Softwarehersteller zur Durchsetzung rigider Urheberrechtsbestimmungen, rät Eltern in einer Presseerklärung, dafür zu sorgen, „dass Ihre Kinder verstanden haben, dass der Download raubkopierter Software nichts anderes ist als Diebstahl" (BSA 2003). Kinder sollen also verstehen, was Moraltheoretiker nicht verstehen können. Weil die Autoren der Evidenz ihrer Argumente dann aber doch nicht vollständig vertrauen, fügen sie noch das klassische Internet-Schreckensszenario für Eltern hinzu: „Stellen Sie sicher, dass sie [die Kinder, U.Z.] sich von Sites fernhalten, die illegale Software anbieten. Diese Sites enthalten oft auch nicht jugendfreies Material wie Pornographie" (ebd.). Die unverfrorenste Demagogie findet sich freilich bei der *Bertelsmann Music Group (BMG)* in deren Kopierschutz-FAQ, die zunächst lapidar behauptet: „Es gibt kein Recht auf Privatkopie" (BMG 2003) und damit eine Minderheitenposition der aktuellen juristischen Diskussion (z.B. Diemar GRUR 2002; dagegen z.B. Knies ZUM 2002, Mayer CR 2003) umstandslos als unumstrittenes Faktum darstellt. Doch kommt es noch zu einer Steigerung: „[Frage] Warum ist denn die Privatkopie überhaupt ‚ausnahmsweise' erlaubt worden? [Antwort] Weil man in den 60er Jahren, als die Ausnahme ins Gesetz kam, keine Möglichkeit sah, ein Verbot technisch durchzusetzen. Der Gesetzgeber wollte die Privatsphäre achten und nicht Millionen Haushalte kriminalisieren. Heute ist technischer Kopierschutz möglich" (BMG 2003). Der diffizile Akt, die oben diskutierte Janusgesichtigkeit „geistigen Eigentums" zwischen finanziellen Anreizen für die Produzenten und dem gesamtgesellschaftlichen Interesse an möglichst ungehinderter Verbreitung von Information auszubalancieren, schnurrt in einer nur mehr als dreist zu bezeichnenden Geschichtsklitterung in das Faktum mangelnder technischer Kontrollmöglichkeiten zusammen. Dass dieses Argument seinen Autoren offenbar glaubwürdig vorkommt, zeigt auf erschreckende Weise, wie hier noch die letzten Überreste eines der gesellschaftlichen Problemlage angemessenen Bewusstseins ideologisch verschluckt wurden.

Natürlich ist es legitim, dass die Software- und Medienindustrie ihre Interessen in die Waagschale wirft, zumindest, soweit sie dabei auf Demagogie verzichtet. Ein *moralisch begründbares Recht* auf „geistiges Eigentum" aber, wie sie glauben machen will, existiert nicht. Als das feudalistische Gesellschaftssystem zerbrach und sich in die Subsysteme bürgerliche Gesellschaft und bürokratisierter Staat ausdifferenzierte, verlor der Adel die systemische Grundlage seiner Existenz und konnte nur noch ein Nischendasein führen, bis er schließlich in der Bedeutungslosigkeit versank. Ein Recht auf den Status quo gab es für ihn damals dennoch genauso wenig.

3. Schillers Utopie

Dass die Handlungslogik des Cyberspace so schwer zu fassen ist, dass es so schwer vorstellbar scheint, die Produktion von Information könne auch ohne das Konstrukt „geistigen Eigentums" im gesellschaftlich erforderlichen Umfang gewährleistet werden, liegt natürlich daran, dass Neues als eben Neues in die überkommenen gesellschaftlichen Begrifflichkeiten sich nicht einfügen will. GNU/Linux und ebenso alle übrige freie Software sind ein gesellschaftliches wie ökonomisches Faktum, der Hintergrund ihrer Entstehung und ihr Ort in der Gesellschaft damit aber noch nicht verstanden. Pekka Himanen beschreibt in seinem Buch *The Hacker Ethic and the Spirit of the Information Age* (Himanen 2001) die freier Software zu Grunde liegende Handlungslogik als intrinsisch motiviert, selbstbestimmt, freiheitlich und solidarisch; das spezifische Merkmal ist dabei klarerweise die intrinsische Motivation. Das allein sagt aber nicht, von woher sich diese Handlungslogik mit der Ausdifferenzierung des Cyberspace Geltung verschafft; hierzu bedarf es eines historischen Blicks.

Die Ausdifferenzierung der vormodernen Gesellschaft in die Subsysteme bürgerliche Gesellschaft und Staat wurde vorbereitet und begleitet von den erheblichen kulturellen Umbrüchen der Aufklärung. Die traditionalen Wertsysteme, in einem langen Prozess innerlich ausgehöhlt, verloren schließlich ihre unhinterfragte Verbindlichkeit und damit ihre gesellschaftsintegrierende Kraft. Im Angesicht suspekt gewordener Dogmatik kommt es zu einer Renaissance der Vernunft, die sich in einem doppelten Rationalisierungsschub hin zu Abstraktion und Formalisierung Bahn bricht: der der erwirtschafteten Sache äußerliche, abstrakte Gewinn wird zum Maß ökonomischer Rationalisierung in der bürgerlichen Gesellschaft, die formale Legitimation des Staates durch Gesellschaftsvertrag und schließlich Wahl zum Maß politischer Rationalität. Der Staat *als* Demokratie legt nur noch das formale Verfahren fest, durch das seine Inhalte generiert werden; keine göttliche Offenbarung gibt seine Ordnung als Spiegel inhaltlicher Werte mehr vor. Der Gewinn *als* alleiniges Ziel ökonomischen Handelns ist unabhängig vom Inhalt der Produktion, mit der er erzielt wurde. Das traditionale Band zwischen Form und Inhalt ist in der Moderne zerrissen; die Ausdifferenzierung der Gesellschaft in zwei Subsysteme ist selbst Ausdruck des Zerfalls übergreifender und integrierender Ordnung.

Dieser Zerfall betrifft genauso die systemische Ebene kulturellen Selbstverständnisses. An Stelle der einen übergreifenden Frage nach dem Gottgewollten treten, in

einer berühmten Formulierung von Immanuel Kant (Kant 1781, S. 833 [Transzendentale Methodenlehre, 2. Hauptstück, 2. Abschnitt]), die drei ausdifferenzierten Fragen „Was kann ich wissen?“, „Was soll ich tun?“ und „Was darf ich hoffen?“, also die eigenständigen Sphären von Wissenschaft/Technik, Moral/Recht und Naturschönem/Kunst oder, in Johann Wolfgang Goethes Friedrich Schiller zugedachten Worten, dem „Wahren, Guten, Schönen“ (Goethe 1805). Und wenn Kant die Frage „Was soll ich tun?“ sodann mit einer – wie oben diskutiert – *formalen* Moraltheorie beantwortet, so ist dies selbst wesentlicher Teil jenes Rationalisierungsschubs der Aufklärung.

Kant hebt damit freilich einen unseligen Dualismus auf eine neue und radikale Ebene, der die westliche Zivilisation seit ihren Wurzeln im antiken Griechenland durchzogen hat. An den Grundfesten unserer Kultur liegt der Sieg des Parmenides über Heraklit in der beiden Streitfrage, ob das Wesen der Welt sich, wie Heraklit meinte, im Wandel oder, wie Parmenides befand, im Unveränderlichen offenbare. Parmenides gewann diesen Disput im Urteil der Antike, und so konnte Platon folgerichtig feststellen, dass alles Physische, da vergänglich und also veränderlich, nur Trugbild sei, und einzig in der Welt des Geistes Wahrheit zu finden wäre, etwa in Form der unveränderlichen Gesetze der Mathematik. Diese strikte erkenntnistheoretische Trennung zwischen Körper und Geist ließ sich weitgehend problemlos abbilden auf den sadomasochistischen Leib-Seele-Dualismus eines ewigkeitsfixierten Christentums, weswegen Platons Texte die finsteren Zeiten des Mittelalters vergleichsweise gut überstehen konnten. Allerdings wusste das Christentum zunächst in seiner traditionalen, integrativen katholischen Form durch ritualisierte Sinnespracht Leib und Seele im Alltag der Gläubigen leidlich gut miteinander zu verbinden. Erst der Protestantismus machte Ernst mit dem Wortlaut der Bibel vom einen, bildnislosen, abstrakten Gott, dem folglich nicht im traditional-opulenten, physisch zelebrierten Ritus, sondern nur im durch die abstrakten Formen der Rationalität vernunftgeleiteten, asketischen Zwiegespräch gerecht zu werden möglich sei.

Mit diesem leibfernen, am rationalisierbaren Ergebnis orientierten Asketismus, der in Folge Arbeit gerade in ihrer Unlust als gottgefällig begreift, ist, wie Max Weber in seiner klassischen soziologischen Untersuchung *Die protestantische Ethik und der Geist des Kapitalismus* (Weber 1904/1905) beschrieben hat, die „geistige“ Basis für die Entfesselung einer ökonomischen Rationalität in der Aufklärung gelegt, die alles Sinnliche jenseits des rational skalierbaren Gewinns tilgt und schließlich die Religion selbst und damit ihre eigenen protestantischen Wurzeln hinter sich lässt. Die Organisation der Arbeit nach den abstrakten Maximen von Rationalisierung und Gewinnmaximierung ist die ökonomisch banalisierte Verfestigung des westlichen Leib-Seele-Dualismus. Diesen Dualismus überhöht Kant transzendentallogisch, wenn er die *Formalität* seiner Moraltheorie so weit treibt, dass ihm aus Angst vor der Wankelmütigkeit (das heißt Vergänglichkeit) sinnlicher Verfassung sich ihr verdankende *Neigungen* zum Hindernis der Moralität gerinnen: nur wer aus (formal deduzierbarer) *Pflicht* handelt, handelt bei Kant moralisch; sobald, was er tut, er auch aus Zuneigung tut, ist dies *keine* moralische Handlung mehr (vergleiche etwa Kant 1785, S. 10f [Akademie-Ausgabe S. 398f]).

Wurde die Aufklärung zunächst als Befreiung vom schwer lastenden Joch einer endlose Jahrhunderte währenden Dogmatik und als erlösender Säkularisierungsprozess erlebt, so macht sich doch bald zunehmendes Unbehagen breit an der nur noch stärker – im rationalen Argument – zementierten Entzweiung von *Körper* und *Geist*, *Leib* und *Seele*, *Inhalt* und *Form*, *Sinnlichkeit* und *Verstand*, *Notwendigkeit* und *Freiheit*, *Neigung* und *Pflicht*. Georg Wilhelm Friedrich Hegel vermeint noch, im Rahmen der Vernunft selbst diese Entzweiung der Moderne mit der philosophischen Idee der (formales Moralgesetz wie traditionale Inhalte umfassenden) *Sittlichkeit* überwinden zu können, die er freilich ausgerechnet im preußischen Obrigkeitsstaat verwirklicht sieht. Danach ist die Philosophie in dieser Frage am Ende; Karl Marx glaubt, sie mit revolutionärer Tat überschreiten zu müssen, und fürderhin ist die gesellschaftliche Abspaltung und Geringschätzung des Leiblichen insbesondere in ihrer Ausprägung als bürgerlicher Unterdrückung der Sexualität ein Thema für Psychoanalytiker und Soziologen.

Als Erster aber hatte sich zuvor Friedrich Schiller explizit dieses Problems angenommen. In seinen Briefen *über die ästhetische Erziehung des Menschen* (Schiller 1795) knüpft er erklärtermaßen an Kants oben diskutierte Moraltheorie an, kritisiert jedoch dessen formalistischen Rigorismus der Pflicht. Der zwischen Neigung und Pflicht hin- und hergerissene Mensch, so Schillers Pointe, ist in seiner als gewaltförmig erfahrenen Zerrissenheit gerade *nicht* frei und somit in der Lage, moralisch zu agieren; erst die *Versöhnung von Pflicht und Neigung* erlaubt wahrhaft freies Handeln und wird damit zur Bedingung der Möglichkeit von Moral (die ja Freiheit als ihre Bedingung der Möglichkeit hat): „Der Wille des Menschen steht aber vollkommen frei zwischen Pflicht und Neigung, und in dieses Majestätsrecht seiner Person kann und darf keine physische Nötigung greifen. Soll er also dieses Vermögen der Wahl beibehalten […], so kann dies nur dadurch bewerkstelligt werden, dass […] seine Triebe mit seiner Vernunft übereinstimmend genug sind, um zu einer universellen Gesetzgebung zu taugen" (Schiller 1795, S. 11 [4. Brief]).

Damit stellt sich die Frage, wie unter den Bedingungen einer in sich zerrissenen Moderne Pflicht und Neigung, mithin Rationalität und Sinnlichkeit, zu versöhnen sind. Schillers Antwort ist die menschliche Praxis des *Spiels*, also dessen, „was weder subjektiv noch objektiv zufällig ist und doch weder äußerlich noch innerlich nötigt" (Schiller 1795, S. 61 [15. Brief]). Spiel entsteht nicht durch *externe Zwänge*, sondern verdankt sich dem *freien Willen* der Phantasie (es „nötigt weder äußerlich noch innerlich"), und ist als solches Ausdruck menschlicher Freiheit (und somit der darin gegründeten Moralität), zugleich aber arbeitet es sich *intentional* an der *determinierten* Welt des Naturgesetzlichen ab (es ist weder „subjektiv noch objektiv zufällig"). Die Integration von Freiheit und Notwendigkeit gelingt dem Spiel gerade dadurch, dass es sich in seiner Freiwilligkeit (als Ausdruck von Freiheit und also Moralität) den *moralischen* Vernunft-„Notwendigkeiten" und in seinem Gestaltungswillen den *instrumentellen* Naturnotwendigkeiten *zugleich* unterwirft, sie aber wechselseitig durch eben diese Freiwilligkeit und eben diesen Gestaltungswillen mit spielerischer Leichtigkeit so zur Deckung zu bringen versucht, dass die Zwänge jedes der beiden Bereiche dem je anderen gegenüber gleichsam ins Leere laufen, weil ihre Forderungen durch die spielerisch erzielte Deckungsgleichheit, die keine

inkongruenten Zufälligkeiten zurücklässt, jeweils bereits erfüllt sind. Die totale Bestimmtheit durch die Zwänge der Vernunft *und* der Natur *zugleich* kippt im Spiel um in die beidseitige, *vollkommene Freiheit*: beide Bereiche werden im Spiel „das Gemüt zugleich moralisch und physisch nötigen; [das Spiel] wird also, weil [es] alle Zufälligkeit aufhebt, auch alle Nötigung aufheben und den Menschen sowohl physisch als moralisch in Freiheit setzen" (Schiller 1795, S. 57 [14. Brief]).

Schiller verwendet zur Illustration dieser Überlegung ein unerwartetes Beispiel, das deutlich macht, wie grundsätzlich bei ihm der Begriff des Spiels die Versöhnung von Pflicht und Neigung mitdenkt: „Wenn wir jemand mit Leidenschaft umfassen, der unsrer Verachtung würdig ist, so empfinden wir peinlich die Nötigung der Natur. Wenn wir gegen einen andern feindlich gesinnt sind, der uns Achtung abnötigt, so empfinden wir peinlich die Nötigung der Vernunft. Sobald er aber zugleich unsre Neigung interessiert und unsre Achtung sich erworben, so verschwindet sowohl der Zwang der Empfindung als der Zwang der Vernunft, und wir fangen an, ihn zu lieben, d.h. zugleich mit unsrer Neigung und mit unsrer Achtung zu spielen" (ebd., S. 57f). Spielen ist die freie Übereinstimmung von *Achtung der Vernunft* (Achtung der Gesetze aus dem Reich der Vernunft wie auch umgekehrt durch die Gesetze der Vernunft – im Falle des Moralischen also der *Pflicht* – angeleitete Achtung) einerseits und *sinnlicher Neigung* andererseits, und das nennen wir im Falle zwischenmenschlicher Beziehungen *Liebe*. *Liebe* oder allgemein *Spiel* wiederum sind, indem sie Neigung und Pflicht versöhnen, wie oben diskutiert, die *Bedingungen der Möglichkeit für Freiheit* und also (einer nunmehr versöhnten) *Moral*. Schiller kann so in einem sehr grundlegenden Sinne formulieren: „Denn, um es endlich auf einmal herauszusagen, der Mensch spielt nur, wo er in voller Bedeutung des Worts Mensch ist, und *er ist nur da ganz Mensch, wo er spielt*" (Schiller 1795, S. 63 [15. Brief]).

Schillers Begriff von Spiel haftet offenkundig nichts Leichtfertiges an; er trägt den ganzen Ernst der Aufgabe, die Wunden einer zerrissenen Moderne zu heilen. Weder geht es um bellizistischen Wettkampf noch um geisttötende Ablenkung. Schillers Spiel ist jener Modus menschlichen Tuns, in dem Kinder in vereinter Vernunft und Sinnlichkeit sich voller Wiss-Begierde die Welt aneignen, und den Erwachsene sich in der Sphäre der *Kunst* erhalten: die *Schönheit* des gelungenen Kunstwerks zeigt sich in der Übereinstimmung von dessen Form und Inhalt und umgreift damit auf seine Weise die in der Moderne entzweiten Pole. Schönheit und Kunst werden zu dem gesellschaftlichen Ort, der Freiheit konstituiert und somit Moral ermöglicht: „Sobald [die Vernunft] demnach den Ausspruch tut: es soll eine Menschheit existieren, so hat sie eben dadurch das Gesetz aufgestellt: es soll eine Schönheit sein" (Schiller 1795, S. 60 [15. Brief]). Unter den Bedingungen der Moderne generieren zwischen den durch eine entfesselte Rationalisierung tief gespaltenen Polen von Vernunft und Natur sowie Pflicht und Neigung erst Spiel und Kunst jene Harmonie, die ein Leben nach menschlichem Maß ermöglicht. Schiller formuliert damit in den *Ästhetischen Briefen* beispielhaft jenen Gedanken der *Humanität*, der neben den Kantischen der Menschenwürde tritt und mit ihm zusammen die Grundwerte umreißt, auf denen unsere Gesellschaft ihrem Selbstverständnis nach

fußt. Seine Überlegungen bilden bis heute die theoretische Folie, auf der sich eine nicht-konservative Modernisierungskritik entfalten konnte.

Somit stellt sich die Frage nach dem gesellschaftlichen Modus der Integrationsleistung von Spiel und Kunst. Eine denkbare Position begreift die Institution Kunst als rezeptiven Ersatz für die in der Aufklärung zerschellten allgemeinverbindlichen religiösen Weltbilder; statt in die Kirche gehen die Menschen in Theater und Konzert. Diesen Ansatz wollte wie kein zweiter Richard Wagner mit der Institution der Bayreuther Festspiele erklärtermaßen umsetzen; seine Idee des *Gesamtkunstwerks* bezeichnet weniger die Gesamtheit der im Musiktheater beanspruchten Künste als vielmehr die gesellschaftliche Integrationsleistung einer solchen Veranstaltung. Die „Erlösung", nach der sich die Protagonisten der von Wagner durch radikale Umdeutung konstruierten „Mythen der Moderne" allesamt sehnen, meint stets die Vereinigung von Rationalität und Sinnlichkeit. In völligem Gegensatz zu dieser bewussten und gezielten Inanspruchnahme ästhetischen Potentials vollbrachte Kunst andererseits gleichsam urwüchsig eine äußerst wirkmächtige Integrationsleistung in den gesellschaftlichen Umbrüchen der sechziger und siebziger Jahre des letzten Jahrhunderts, die ohne die neue, die technischen Artefakte des Zeitalters klanglich transformierende Sinnlichkeit der Rockmusik in solcher Form schlechterdings nicht denkbar gewesen wären. Doch obwohl diese beiden Kunstereignisse auf je ihre Weise so erfolgreich wie nur überhaupt vorstellbar waren, vermochten sie gleichwohl nicht, das Versprechen des Schiller'schen Spiels wirklich einzulösen. Die Bayreuther Festspiele sind in ihrer zur Institution gewordenen Dauerhaftigkeit eines künstlerischen Ausdrucks in gesellschaftsintegrativer Absicht gewiss eine weltweit einzigartige kulturelle Errungenschaft, und dennoch wird sich kaum behaupten lassen, dass sie über den Bereich der Kunst selbst hinaus in relevantem Umfang die Gesellschaft geprägt haben. Die Rockmusik der sechziger und siebziger Jahre andererseits hat wie nie eine künstlerische Entwicklung zuvor über den halben Erdball hinweg gesellschaftliche Entwicklungen integriert, war aber nur von kurzer Dauer, bis sie in weiten Teilen zum Industrieprodukt degenerierte und ihre umfassende gesellschaftliche Funktion einbüßte beziehungsweise, schlimmer, in betäubende Affirmation moderner Pathologien transformierte. Beide Kunstereignisse blieben in ihrer gesellschaftlichen Wirkung begrenzt, im einen Fall räumlich, im anderen zeitlich.

Das hat seinen Grund darin, dass Kunst als (rezeptive) Institution auf der systemischen Ebene kultureller Selbstdeutung angesiedelt ist und nicht auf der Ebene der Vergesellschaftung wie bürgerliche Gesellschaft und Staat. Der zwar nicht Ge-, wohl aber Missbrauch der Bayreuther Festspiele durch den Nationalsozialismus zeigt freilich in aller Deutlichkeit, dass eine kurzschlüssige Ästhetisierung des vergesellschaftenden Subsystems „Staat" selbst keine Alternative ist, die in die Freiheit führt, sondern die im Gegenteil in der Barbarei enden kann. Die Ästhetisierung des Tauschs der bürgerlichen Gesellschaft wiederum ist in Form der Werbung zwar allgegenwärtig, aber ebenfalls schwerlich versöhnende Kunst, sondern unter dem Schleier abgerichteter Sinnlichkeit vielmehr eine weitere Verfestigung der abstrakten Rationalität des Gewinns. Schiller warnt ausdrücklich vor der Vermengung künstlerischer, ökonomischer und politischer Handlungslogiken; der schöne Schein der Kunst wird in Politik und Ökonomie zum niedrigen Schein der Heuchelei und Täu-

schung: „Nur soweit er *aufrichtig* ist (sich von allem Anspruch auf Realität ausdrücklich lossagt), und nur soweit er *selbständig* ist (allen Beistand der Realität entbehrt), ist der Schein ästhetisch. Sobald er falsch ist und Realität heuchelt, und sobald er unrein und der Realität zu seiner Wirkung bedürftig ist, ist er nichts als ein niedriges Werkzeug zu materiellen Zwecken und kann nichts für die Freiheit des Geistes beweisen" (Schiller 1795, S. 115 [26. Brief]).

Kunst ist also *nicht* der Ort, an dem Spiel zum vergesellschaftenden Prinzip werden kann, sondern bleibt ein Verfahren kultureller Selbstverständigung und -deutung. Jürgen Habermas hat daraus gefolgert, dass Schiller sich mit diesem Wirkungskreis seiner Überlegungen bescheidet: „Schillers ästhetische Utopie zielt freilich nicht auf eine Ästhetisierung der Lebensverhältnisse, sondern auf eine Revolutionierung der Verständigungsverhältnisse" (Habermas 1985, S. 63). Doch Schiller selbst widmet den gesamten letzten Brief ausdrücklich der Frage nach dem *ästhetischen Staat*, einem Staat also, in dem das Spiel sehr wohl – wenn auch nicht in Form der Kunst – ein Modus der Vergesellschaftung wird; in dem der Mensch dem Menschen „nur als Objekt des freien Spiels gegenübersteh[t]" (Schiller 1795, S. 125 [27. Brief]). Freilich fragt sich Schiller am Ende seiner Abhandlung selbst: „Existiert aber auch ein solcher Staat des schönen Scheins, und wo ist er zu finden?" (Schiller 1795, S. 128 [27. Brief]). Einen Hinweis darauf, wie solch ein Staat beschaffen sein müsste, hatte Schiller zuvor bereits gegeben: „[…] solange die Not gebietet und das Bedürfnis drängt, ist die Einbildungskraft mit strengen Fesseln an das Wirkliche gebunden; erst wenn das Bedürfnis gestillt ist, entwickelt sie ihr ungebundenes Vermögen" (Schiller 1795, S. 112 [26. Brief]). Unter ständigem Druck knapper Ressourcen kann spielerische Phantasie sich nicht entfalten.

Dies ist der entscheidende Fingerzeig zur gesellschaftlichen Verortung des Schiller'schen Spiels. Radikaler als Kunst, die als kulturelles Deutungsmuster nicht auf die systemische Ebene der Vergesellschaftung geholt, sondern auf dieser Ebene nur, darin den traditionalen religiösen Weltbildern gleich, *rezipiert* werden kann, wartet das Spiel darauf, *in der Gesellschaft* gespielt zu werden, um dort Pflicht und Neigung zu versöhnen. Es bedarf dazu jedoch eines gesellschaftlichen Systems ohne Mangel, und dieses System hat es bislang nicht gegeben; Kunst war alles, was von Schillers ästhetischen Idealen Wirklichkeit beanspruchen konnte. Mit dem Cyberspace aber differenziert sich ein drittes vergesellschaftendes Subsystem aus, das keinen Mangel an seinen virtuellen, beliebig reproduzierbaren Gütern kennt. Angesichts seiner distributiven Logik der Partizipation kann das Schiller'sche Spiel hier gespielt werden.

Pekka Himanen kontrastiert in seinem zu Beginn dieses Kapitels erwähnten Buch die *Hacker Ethic* immer wieder mit der zu ihr im scharfen Gegensatz stehenden protestantischen Ethik, wie sie Max Weber als kennzeichnend für kapitalistische Ökonomien herausgearbeitet hat. Aber Himanen zeigt nicht, woher jene neue Ethik intrinsischer Motivation kommt. Diese Frage lässt sich mit Schillers Überlegungen nun beantworten. Sein Programm der Versöhnung von Pflicht und Neigung, das in den dualistisch zerrissenen Grundfesten der Moderne wurzelt und seit Beginn der Aufklärung seiner Verwirklichung harrt, ist eine anspruchsvolle Theorie intrinsisch motivierten Handelns. Mit der Aufklärung hatte sich die systemische Ebene kultu-

reller Selbstdeutung in die drei Wertsphären von Wissenschaft, Moral und Kunst – dem Wahren, Guten und Schönen – ausdifferenziert, die systemische Ebene der Vergesellschaftung hingegen nur in die beiden Subsysteme von bürgerlicher Gesellschaft und Staat. Wissenschaft formt die technikgestützte Industrieproduktion der bürgerlichen Gesellschaft; Moral findet seine Ausformung in den Mechanismen des Rechtsstaates. Es hat den Anschein, als harre die verbleibende Wertsphäre der Kunst eines ihr korrelierenden, intrinsisch bestimmten dritten Subsystems der Vergesellschaftung, das einer weiteren gesellschaftlichen Ausdifferenzierung durch eine zweite technologische Revolution entspränge, und als wäre der Cyberspace eben dieses Subsystem.

Die dem Cyberspace eigene Ökonomie der Partizipation erlaubt es dem Schiller'schen Spiel erstmals, vergesellschaftend zu wirken; intrinsisch motiviertes Handeln steht im Zentrum dieser Ökonomie. Linus Torvalds vertritt in seinem Prolog zu Himanens Buch die (in der, wie er selbst schreibt, ihm eigenen Bescheidenheit „Linus's Law" betitelte) These, jedwede Motivation menschlichen Handelns würde in eine von drei Kategorien fallen: *survival, social life* und *entertainment* (Torvalds 2001, S. xiv) – die Parallelität zu Wissenschaft/Technik, Moral und Kunst/Spiel springt ins Auge. Die motivationalen Grundlagen der Open-Source-Bewegung sieht Torvalds dabei im *entertainment*. Schiller betont, dass *Spiel* in seinen Überlegungen nicht in einem billigen Sinne zu verstehen sei: „Freilich dürfen wir uns hier nicht an die Spiele erinnern, die in dem wirklichen Leben im Gange sind und die sich gewöhnlich nur auf sehr materielle Gegenstände richten" (Schiller 1795, S. 62 [15. Brief]). Ganz entsprechend argumentiert Torvalds: „Entertainment may sound like a strange choice, but I mean by *entertainment* more than just playing games on your Nintendo. It's chess. It's painting. It's the mental gymnastics involved in trying to explain the universe. Einstein wasn't motivated by survival when he was thinking about physics. Nor was it probably very social. It was entertainment to him. Entertainment is something intrinsically interesting and challenging" (Torvalds 2001, S. xv). Die Ökonomie der Partizipation jenseits „geistigen Eigentums", die mit dergestalt intrinsisch motivierter Produktion verwoben und die die Grundidee freier Software ist, beschreibt bereits Schiller als kennzeichnenden Bestandteil der Welt des Spiels (die in Schillers Terminologie von „Geschmack" geleitet wird): „Aus den Mysterien der Wissenschaft führt der Geschmack die Erkenntnis unter den offenen Himmel des Gemeinsinns heraus und verwandelt das Eigentum der Schulen in ein Gemeingut der ganzen menschlichen Gesellschaft" (Schiller 1795, S. 127 [27. Brief]). Und auch die Produktionsbedingungen in der Welt des Spiels charakterisiert Schiller in Übereinstimmung mit Himanen. Himanen schildert den unter den Maximen der *Hacker Ethic* ebenso selbstbestimmten wie vollständigen Rückzug zu konzentrierter Arbeit: „When Torvalds programmed his first versions of Linux, he typically worked late into the night and then woke up in the early afternoon to continue" (Himanen 2001, S. 20). Zugleich betont Himanen die spontane weltweite Kommunikation übers Internet, sobald Hilfe benötigt wird: „[Torvalds] has never hesitated to ask for help" (ebd., S. 74), Hilfe, die er sogleich in der „Net Academy" (ebd., S. 75) des weltumspannenden Netzes findet, dem er schließlich das Ergebnis seiner Arbeit mitteilt. Bei Schiller entwickelt sich spielerische Produktivität „da al-

lein, wo [der Mensch] in eigener Hütte still mit sich selbst und, sobald er heraustritt, mit dem ganzen Geschlechte spricht" (Schiller 1795, S. 111 [26. Brief]).

Kritiker der Open-Source-Bewegung verweisen bisweilen mit einer gewissen Häme darauf, dass der Erstellungsprozess von freier Software mitnichten demokratischer Natur sei – es gebe sehr wohl tonangebende Figuren. Die Suggestion, Vertreter der Open-Source-Bewegung würden eine demokratische Verfahrenslogik für sich beanspruchen, zeugt freilich von Konfusion bezüglich des gesellschaftlichen Orts freier Software; sie ist nicht im politischen Subsystem des Staates, sondern im partizipativen des Cyberspace angesiedelt; „Demokratie" ist jedoch eine politische Kategorie. Die Open-Source-Bewegung kennt sehr wohl Hierarchien, aber *nicht* die der politischen Macht oder des Gewinns aus Tauschgeschäften; verwandt ist ihr in ihrer Intrinsik am ehesten die wenn überhaupt, dann „meritokratische" Logik künstlerischen Handelns. Gesellschaftlich bedeutsam ist die Open-Source-Bewegung freilich gleichwohl: denn im Gegensatz zur Kunst siedelt sie in einem *vergesellschaftenden* Subsystem, dem neu ausdifferenzierten Subsystem des Cyberspace. Die Schiller'sche Vision des Spiels als einer intrinsischen Praxis, die eine ästhetische, von den dualistischen Pathologien der Moderne befreite Gesellschaft zu konstituieren vermag, hat damit das erste Mal seit ihrer Formulierung in der Aufklärung die Chance, real zu werden. Der Cyberspace wäre dann keine Laune der Zeit, sondern die Vollendung der technologisch getragenen Ausdifferenzierung der Moderne, die über 200 Jahre auf sich warten ließ. Er wäre das gesellschaftliche Subsystem, in dem sich Humanität erstmals, wie bescheiden auch immer, in den Prozessen der Vergesellschaftung verankern könnte und damit eine intrinsisch versöhnte, humane Ökonomie ermöglichen würde, die den unseligen Leib-Seele-Dualismus unserer Kultur nicht mehr als von den Gegenständen der Produktion abstrahierte Rationalisierung und Gewinnmaximierung perpetuierte.

4. Was sollen wir tun?

Aus der Perspektive politischer Steuerung betrachtet ist die Debatte um die aufkommende Partizipationsgesellschaft und die Handlungslogik des Cyberspace zumindest im Moment, da die Kräfte des Status quo noch energisch versuchen, die neuen Entwicklungen unter das Bestehende zu fügen, wesentlich eine Debatte um „geistiges Eigentum" in seinen rechtlichen Ausformungen von Urheberrecht und Patentschutz. Dass „geistiges Eigentum" ein pragmatisches Konstrukt ist, auf das es *keinen moralischen* Anspruch gibt, wurde oben ausführlich diskutiert. Mehr noch: aus rein moraltheoretischer Sicht ließe sich heute wenn, dann nur argumentieren, dass im Cyberspace die freie Weitergabe jeglicher Information sogar *geboten* wäre, nicht zuletzt um einer abgesicherten demokratischen Öffentlichkeit und, aus Kantischer Perspektive, einer umfassenden Geltung der Menschenwürde willen.

Politisch freilich ist zu berücksichtigen, dass angesichts des gerade erst anbrechenden Informationszeitalters und angesichts des empirischen Dunkels über das Zusammenspiel der Ökonomien von Cyberspace und bürgerlicher Gesellschaft, „geistiges Eigentum" als gesellschaftlich etablierter Mittler zwischen beiden Ökonomien sicher nur schrittweise, in einem evolutionären Prozess, zurückgenommen

werden kann. Lawrence Lessig unterbreitet in *The Future of Ideas* hierzu etliche Vorschläge, von der zeitlichen Limitierung urheberrechtlicher Ansprüche auf wenige Jahre bis zur generellen Freigabe nichtkommerziellen Kopierens (Lessig 2001, S. 250f).

Momentan geht die Entwicklung allerdings mit einer eigentümlichen Selbstverständlichkeit, so, als müsse schlicht eine löchrige Tonne abgedichtet werden, in die genau entgegengesetzte Richtung, zu immer größerer Ausdehnung der Bereiche, die „geistigem Eigentum" untergeordnet werden. Es ist daher von um so größerer Bedeutung, sich zu vergegenwärtigen, dass „geistiges Eigentum" nichts ist, was von uns einzufordern wer auch immer ein moralisches Anrecht hätte, *keine* spezielle Form von materialem Eigentum, sondern ein pragmatisches Konstrukt in *Analogie* zu materialem Eigentum. Es obliegt *unserer freien Entscheidung*, welchen Werten wir mit diesem Konstrukt Rechnung tragen wollen.

Und an dieser Stelle greift Lessig zu kurz. Möglicherweise aus argumentationsstrategisch guten Gründen konzentriert er sich auf das bürgerlich-ökonomische Argument, dass Internet und freie Software einen gewaltigen wirtschaftlichen Innovationsschub ausgelöst hätten, der unter der immer weiteren Ausdehnung „geistigen Eigentums" nun zu ersticken drohe. Das ist sicher richtig und wichtig, aber es geht um mehr.

Die Open-Source-Bewegung ist ein starkes empirisches Indiz dafür, dass sich mit dem Cyberspace neben bürgerlicher Gesellschaft und Staat ein neues vergesellschaftendes Subsystem mit eigener Handlungslogik herausbildet; eine Ökonomie der Partizipation und intrinsisch motiviertes Handeln stehen im Mittelpunkt dieses Subsystems. Der an Friedrich Schiller geschärfte Blick enthüllt, dass damit erstmals Grundwerte der Moderne in ökonomisch relevanter Gestalt zum Tragen kommen, auf denen das klassische Ideal der Humanität sich zu seinem vollen gesellschaftlichen Begriff zu entfalten eine Möglichkeit hätte.

Freilich ist der Cyberspace nur Teil der Gesellschaft. Wir wissen aber aus früheren Epochen, dass die für die Epoche paradigmatische Handlungslogik auf die Gesellschaft insgesamt ausstrahlt. Der Handlungslogik des Cyberspace Raum zu geben und seine Ökonomie der Partizipation nicht gewaltsam der bürgerlichen Ökonomie des Tausches einzugliedern, hieße daher, der Informationsgesellschaft eine Chance einzuräumen, Humanität in Wirtschaft und Gesellschaft in einem der Moderne gemäßen, bislang aber nicht realisierten Umfang auszubilden.

Das ist es, was wir gegen die eigentumsrechtlichen Interessen der Medien- und Softwareindustrie abzuwägen haben.

Literaturverzeichnis

Adobe (2003): *Ausführliche Informationen zur Aktivierung von Adobe Photoshop CS*, online http://www.adobe.de/activation/main.html (29.9.2003).

Balzli, Beat / Kerbusk, Klaus-Peter / Rosenbach, Marcel / Schulz, Thomas (2003): *Alles nur geklaut*, Der Spiegel 36/2003, 72, online http://www.spiegel.de/spiegel/0,1518,263671,00.html (29.9.2003).

Bentham, Jeremy (1789): *An Introduction to the Principles of Morals and Legislation*, London ²1823,
online http://www.econlib.org/library/Bentham/bnthPMLtoc.html (29. 9. 2003).

BMG (2003): *Kopierschutz bei BMG: FAQs*,
online http://www.bmg.de/stuff/kopierschutz/index2.html (29. 9. 2003).

BSA (2003): *Drei Jugendliche wegen Raubkopierens belangt*,
online http://global.bsa.org/germany/presse/aktuell/091202.php (29. 9. 2003).

Diemar, Undine von (2002), *Kein Recht auf Privatkopien – Zur Rechtsnatur der gesetzlichen Lizenz zu Gunsten der Privatvervielfältigung*, GRUR (Gewerblicher Rechtsschutz und Urheberrecht) 7/2002, S. 587 ff.

Goethe, Johann Wolfgang (1805): *Epilog zu Schillers Glocke*, in: Trunz, Erich (Hg.): Werke I. Hamburger Ausgabe, München 1982, S. 257 ff.
online http://gutenberg.spiegel.de/goethe/gedichte/glocke.htm (29. 9. 2003).

Habermas, Jürgen (1983): *Moralbewusstsein und kommunikatives Handeln*. Frankfurt am Main.

Habermas, Jürgen (1985): *Der philosophische Diskurs der Moderne. Zwölf Vorlesungen*, Frankfurt am Main.

Habermas, Jürgen (1992): *Faktizität und Geltung. Beiträge zur Diskurstheorie des Rechts und des demokratischen Rechtsstaats*, Frankfurt am Main.

Himanen, Pekka (2001): *The Hacker Ethic and the Spirit of the Information Age*, New York/Toronto.

Kant, Immanuel (1781): *Kritik der reinen Vernunft*, Riga ²1787,
online http://gutenberg.spiegel.de/kant/krvb/krvb.htm (29. 9. 2003).

Kant, Immanuel (1785): *Grundlegung zur Metaphysik der Sitten*, Riga ²1786.

Kant, Immanuel (1788): *Kritik der praktischen Vernunft*, Riga 1788.
online http://gutenberg.spiegel.de/kant/kritikpr/kritikpr.htm (29. 9. 2003).

Knies, Bernhard (2002): *Kopierschutz für Audio-CDs. Gibt es einen Anspruch auf Privatkopie?* ZUM (Zeitschrift für Urheber- und Medienrecht) 11/2002, S. 793.

Lessig, Lawrence (2001): *The Future of Ideas. The Fate of the Commons in a Connected World*, New York.

Mayer, Christoph (2003): *Die Privatkopie nach Umsetzung des Regierungsentwurfs zur Regelung des Urheberrechts in der Informationsgesellschaft. Verkommt der Begriff „Recht zur Privatkopie" zum bloßen Euphemismus?* CR (Computer und Recht) 4/2003, S. 274 ff.

Pareto, Vilfredo (1909): *Manuel d'economie politique*, Paris.

Rawls, John (1971): *A Theory of Justice*, Cambridge Massachusetts.

Schiller, Friedrich (1795): *Über die ästhetische Erziehung des Menschen, in einer Reihe von Briefen*, Stuttgart 1965.
online http://gutenberg.spiegel.de/schiller/aesterz/aesterz.htm (29. 9. 2003).

Torvalds, Linus (2001): *What Makes Hackers Tick? a.k.a. Linus's Law*, in: Himanen (2001), S. xiii ff.

Weber, Max (1904/1905): *Die protestantische Ethik und der Geist des Kapitalismus*, in: ders.: Gesammelte Aufsätze zur Religionssoziologie I, Tübingen 1920.

Zappe, Uli (2003): *Das Ende des Konsums in der Informationsgesellschaft*, in: Fischer, Peter / Hubig, Christoph / Koslowski, Peter (Hg.): Wirtschaftsethische Fragen der E-Economy, Heidelberg 2003, S. 48 ff.
online http://www.ritual.org/Herbst/Konsumende.pdf (29.9.2003).

Anhang

Autorenverzeichnis

Eva Brucherseifer, Dipl.-Ing., war seit Dez. 1998 nach dem Studium der Elektrotechnik an der Technischen Universität Darmstadt als wissenschaftliche Mitarbeiterin am Fachgebiet Regelungstheorie & Robotik an der TU Darmstadt tätig. Seit Herbst 2000 arbeitet sie am KDE-Projekt mit; sie organisierte u.a. mehrere Messestände. Seit August 2002 ist sie stellvertretende Vorsitzende des KDE e.V. Im Mai 2003 gründete sie die Firma *Basyskom Ingenieurbüro Brucherseifer und Ackermann GbR*, die Auftragsentwicklungen für Business-Anwendungen sowie Dienstleistungen im KDE-Umfeld anbietet.

Roman Büttner studiert an der TU Berlin Neuere und Mittelalterliche Geschichte sowie Informatik mit dem Abschlussziel des Magister Artium. Im Rahmen des Studiums beschäftigte er sich ausführlich mit Medien und Kommunikation sowie deren Auswirkungen auf die Gesellschaft. Im Redaktionsteam dieses Jahrbuches übernahm er die Betreuung des Kapitels „Gesellschaft“ und die Gestaltung des Text-Layouts.

Steven Clift ist ein Online-Aktivist und Redner mit Schwerpunkt auf Nutzung des Internets für Demokratie, Governance und Gesellschaft. Im letzten Jahrzehnt konzentrierte er seine Anstrengungen auf die grundlegende Verbesserung der Demokratie und Bürgernähe durch das Internet. Als einer der führenden Experten auf dem Gebiet der E-Democracy versucht er, aktiv Menschen auf der ganzen Welt miteinander in Verbindung zu bringen, um mit diesem neuen Medium etwas zu bewegen. 2002 erhielt er den Minneapolis Award. 2001 und 2003 war er als einer der „25 die die Welt des Internets und der Politik verändern“ von „Politics Online“ aufgeführt. Drei Jahre lang koordinierte er die Anstrengungen von Minnesotas Regierung sich im Internet zu präsentieren. Außerdem war er Co-Autor der „G8 Democracy and Government Online Services“-Publikation (1999).

Matthias Ettrich ist Diplom-Informatiker und seit 8 Jahren als Entwickler Freier Software aktiv. Als Gründer des K-Desktop-Environment-(KDE-)Projektes konnte er miterleben, wie aus einer Idee und einer kleinen Gruppe von Leuten ein internationales Projekt mit über tausend freiwilligen Mitarbeitern wurde. Beruflich arbeitet Matthias als Leiter der Entwicklung bei einem kommerziellen Softwarehersteller im Open-Source-Umfeld.

Peter H. Ganten ist Gründer und Geschäftsführer der Univention GmbH. Er hat sich mehrfach an der Entwicklung von Open-Source-Software beteiligt und arbeitete vor der Gründung der Univention GmbH als IT-Consultant, Trainer und

Autor. Mit der Univention GmbH war er an zahlreichen Migrationsprojekten beteiligt.

Yacine Gasmi studierte zunächst Medieninformatik an der Eberhardt-Karls-Universität Tübingen, bevor er zum Hauptstudium an die TU-Berlin wechselte, wo er sich seit Oktober 2002 innerhalb des Studiengebietes „Informatik und Gesellschaft" mit gesellschaftspolitischen Aspekten der Informatik befasst. Im Jahrbuch war er verantwortlich für das Kapitel „Technik" und die Gestaltung der Internetpräsenz.

Robert A. Gehring, Dipl. Inform, studierte Elektrotechnik, Informatik und Philosophie an der Technischen Hochschule Ilmenau und an der Technischen Universität Berlin. Nach dem Studium arbeitete er freiberuflich als Consultant, Dozent und Autor, bevor er an die TU Berlin zurückkehrte. Dort arbeitet er im Fachgebiet Informatik und Gesellschaft als wissenschaftlicher Mitarbeiter mit den Forschungsschwerpunkten Open Source, IT-Sicherheit und „geistiges Eigentum". Er promoviert zu Fragen der Softwareökonomie und des Softwarerechts. Neben der Wahrnehmung der Aufgaben als Herausgeber hat Robert Gehring für dieses Jahrbuch einen Aufsatz zum Problem der Sicherheit von Open Source beigesteuert.

Lydia Heller, M.A., ist Kommunikationswissenschaftlerin und Journalistin. Sie studierte Kommunikations-, Wirtschafts- und Politikwissenschaften an der Freien Universität Berlin. Gegenwärtig ist sie Volontärin bei der Deutschen Welle in Bonn.

Ursula Holtgrewe, PD Dr. rer. pol., ist Soziologin und lehrt Arbeits- und Organisationssoziologie an der Universität Duisburg/Essen. Sie hat Soziologie, Politikwissenschaften, Europäische Ethnologie, Neuere Deutsche Literatur und Communication Studies in Marburg und London studiert, 1996 an der Universität Kassel promoviert und sich 2003 an der Universität Duisburg/Essen habilitiert. Ihre Arbeitsgebiete sind Organisation, Innovation und Subjektivität sowie Dienstleistungsarbeit und Handlungstheorie.

Andreas John, begann nach abgeschlossener Ausbildung zum Bankkaufmann das Studium der Informatik an der TU Berlin. Seine Studienschwerpunkte liegen im Bereich Informatik und Gesellschaft sowie Softwaretechnik. Innerhalb dieses Jahrbuches betreute das Kapitel Ökonomie und die Internetpräsenz.

Svetlana Kharitoniouk, geboren in Weißrussland und in Moskau/Russland aufgewachsen, kam 1996 nach Berlin um eine Ausbildung zur Industriekauffrau aufzunehmen. Nach dem erfolgreichen Abschluss hat sie das Studium des Wirtschaftsingenieurwesens mit dem technischen Schwerpunkt „Informations- und Kommunikationssysteme" an der TU Berlin aufgenommen. Im Redaktionsteam übernahm sie neben der Betreuung des Grundlagen-Kapitels (Kapitel 1) die Projektleitung und die Koordination der Marketingaktivitäten.

Olaf Koglin studierte nach einer kaufmännischen Ausbildung Rechtswissenschaften sowie Politik und promovierte zu Rechtsfragen von Open-Source-Software an der Universität Bonn. Anschließend war er als Rechtsanwalt in IT-rechtlichen Kanzleien in Deutschland und den USA tätig. Er ist ständiger Mitarbeiter des Instituts für Rechtsfragen der Freien und Open-Source-Software (ifrOSS). Seit 2003 arbeitet er in der Kanzlei „Lenhardt Rechtsanwälte".

Dirk Kuhlmann studierte Informatik an der Technischen Universität Berlin und ist seit 1995 im Bereich Computersicherheit für die Hewlett Packard Laboratories in Bristol (UK) tätig. Er hat in den Bereichen Echtzeitsysteme, elektronisches Publizieren und sichere Transaktionen für Finanzdaten geforscht. Zurzeit befasst er sich vornehmlich mit theoretischen und praktischen Aspekten von *Trusted Computing*. Sein besonderes Interesse gilt der Sicherheit und Vertrauenswürdigkeit von Open-Source-basierten Systemen.

Uwe Küster, Dr., MdB, studierte in Magdeburg Physik und promovierte 1964 dort. Von 1969–1991 war er als wissenschaftlicher Mitarbeiter an der Medizinischen Akademie Magdeburg tätig, wo er sich 1982 habilitierte. Sein im Herbst 1989 begonnenes politisches Engagement führte ihn 1990 in die SPD, für die er seitdem als Abgeordneter in den Deutschen Bundestag gewählt ist. Dort ist er neben vielen anderen Funktionen parlamentarischer Geschäftsführer der SPD-Fraktion, Mitglied des Ältestenrates und Sprecher der SPD-Fraktion in der Kommission des Ältestenrates für den Einsatz neuer Informations- und Kommunikationstechniken und -medien.

Bernhard Kuster, Lic. Phil., arbeitet seit seinem Abschluss des Studiums der Soziologie an der Universität Zürich als Forschungsassistent am Lehrstuhl von Frau Prof. Dr. M. Osterloh. Zur Zeit schreibt er seine Dissertation über Open-Source-Software. Bernhard Kuster ist dabei vor allem daran interessiert, welches die Gründe für den Erfolg von Open-Source-Software sind und inwiefern man daraus allgemeine theoretische Erkenntnisse über Innovation gewinnen kann.

Raphael Leiteritz, Dipl. Inform., Cand. MBA INSEAD, studierte an der Technischen Universität Berlin Informatik. Er ist ehemaliger Vorstand des Linux-Unternehmens Innominate, das er während des Studiums gründete. Zur Beendigung des Studiums verließ er die Firma und arbeitete nach Abschluss des Diploms als freier wissenschaftlicher Mitarbeiter im Forschungsgebiet Informatik und Gesellschaft an der TU Berlin mit. Gegenwärtig absolviert er ein MBA-Studium am INSEAD in Frankreich und Singapur.

Benno Luthiger hat an der Universität Zürich theoretische Physik studiert (Abschluss 1988) und 1997 ein zweites Studium in Ethnologie abgeschlossen. Parallel dazu arbeitete er etliche Jahre als Software-Entwickler. Seit 2001 befasst er sich intensiv mit Open-Source-Software. Im Rahmen des Forschungsprojekts FASD am

Institut für betriebswirtschaftliche Forschung der Universität Zürich untersucht er die theoretischen Aspekte dieses Software-Entwicklungsmodells. Als Fachperson für Open Source der Informatikdienste der ETH Zürich beschäftigt er sich mit den technologischen Fragen von quelloffener Software.

Bernd Lutterbeck, Prof. Dr. iur., studierte Rechtswissenschaften und Betriebswirtschaftslehre in Kiel und Tübingen. Danach folgten wissenschaftliche Tätigkeiten an den Universitäten Regensburg (1969–1971, Fachbereich Rechtswissenschaft) und Hamburg (1974–1978, Dozent am Fachbereich Informatik) sowie 1976 die Promotion zum Dr. iur. an der Universität Regensburg. Von 1978–1984 war er Beamter beim Bundesbeauftragten für den Datenschutz in Bonn. Seit 1984 ist Bernd Lutterbeck Professor für Wirtschaftsinformatik an der Technischen Universität Berlin mit den Schwerpunkten Informatik und Gesellschaft, Datenschutz- und Informationsrecht, Verwaltungsinformatik. Seit 1995 nimmt er für die Action Jean Monnet der Europäischen Union Brüssel an der Technischen Universität Berlin eine Professur für humanwissenschaftliche Fragen der europäischen Integration wahr. Seine aktuelle Arbeitsschwerpunkte sind E-Government (zusammen mit dem Bundesminister des Innern), Theorie und Praxis der *property rights*, Aufbau einer Open-Source-Software-Umgebung, *European Governance*. Er ist einer der Herausgeber dieses Jahrbuchs.

Axel Metzger, Dr. iur., studierte Rechtswissenschaften in Hamburg, Paris und München mit Schwerpunkt „Medienrecht". 1999-2000 promovierte er am Max-Planck-Institut für ausländisches und internationales Patent-, Urheber- und Wettbewerbsrecht in München, 2002 legte er das Assessorexamen in Hamburg ab. Seit 2002 ist er wissenschaftlicher Referent am Max-Planck-Institut für ausländisches und internationales Privatrecht in Hamburg. Axel Metzger ist Mitbegründer und Leiter des Instituts für Rechtsfragen der Open-Source-Software (ifrOSS).

Frank Müller, hat vor seinem Studium an der HAB Weimar und an der Technischen Hochschule Wismar eine Berufsausbildung als Baufacharbeiter absolviert. Bevor er 1996 seine Tätigkeit als IT-Referent im Landesrechnungshof Mecklenburg-Vorpommern aufnahm, hat er berufliche Erfahrungen als Mitarbeiter einer Stadtverwaltung, als ADV-Verantwortlicher im Finanzamt und als IT-Consultant bei einem Systemhaus gesammelt. Zu seinen Qualifikationen im IT-Bereich zählen u. a.: *Microsoft Certified Systems Engineer*, *Microsoft Certified Professional*, *ITIL-Foundation Certificate*, verschiedene Betriebssystemlehrgänge im Unix-, Novell- und NT-Umfeld.

Sabine Nuss ist Politologin und Journalistin. Sie studierte an der FU Berlin Politikwissenschaften im Diplomstudiengang und promoviert derzeit zum Thema „Privateigentum in einer immateriellen Welt." Sie ist Redaktionsmitglied der Prokla, Zeitschrift für kritische Sozialwissenschaft.

Margit Osterloh, Prof. Dr., ist ordentliche Professorin für Betriebswirtschaftslehre, insbesondere Organisationslehre, am Institut für betriebswirtschaftliche For-

schung der Universität Zürich. Die Forschungsschwerpunkte von Margit Osterloh sind Organisationstheorie, Innovations- und Technologiemanagement und Frauen in der Unternehmung.

Gregorio Robles ist Assistent und Doktorand an der Universidad Rey Juan Carlos in Madrid. Der Schwerpunkt seiner Forschungsarbeit liegt auf der Untersuchung freier Softwareentwicklung aus softwaretechnischer Sicht mit besonderem Bezug auf quantitative Methoden. Er war beteiligt am Entwurf von Programmen zur Automatisierung der Analyse von Freier Software sowie der zu ihrer Herstellung verwendeten Softwarewerkzeuge. Gregorio Robles war an der vom IST-Programm der EU-Kommission finanzierten FLOSS-Studie beteiligt.

Sandra Rota, Lic. oec. Publ., ist seit 1999 Doktorandin und Forschungsassistentin am Lehrstuhl für Organisation, Technologie- und Innovationsmanagement von Frau Prof. Dr. M. Osterloh an der Universität Zürich. In ihrer Forschung befasst sie sich mit der Rolle von intellektuellen Eigentumsrechten in kooperativen Innovationsprozessen. Ihre Arbeiten zum Thema Open Source beschäftigen sich hauptsächlich mit der Frage, unter welchen Bedingungen kollektive Innovationsprozesse ohne die Zuteilung individueller intellektueller Eigentumsrechte auskommen.

Hendrik Scheider studiert Informatik sowie systematische Sprachwissenschaft und Mediensemiotik an der TU Berlin. Er hat an der weltweit beachteten Studie „Who Is Doing It?" zur Motivation von Libre-Software-Entwicklern mitgearbeitet. Seine Diplomarbeit trägt den Titel „Eine Ausführungsumgebung für Mobile Java im Rahmen interaktiver Gerätesimulationen." Als Software-Entwickler arbeitet er in einer auf Simulationen und Fragen der Handhabbarkeit mobiler Endgeräte spezialisierten Firma.

Alfred Schröder, Dipl. Wirt. Inform., ist Geschäftsführer der GONICUS GmbH und war Projektleiter bei der OSS-Migration des Instituts für Tierzucht der Bundesforschungsanstalt für Landwirtschaft. Nach dem Studium der Wirtschaftsinformatik und den ersten Kontakten mit GNU/Linux und Open-Source-Software begann er seinen beruflichen Werdegang bei einem kleinen Beratungsunternehmen und Internet-Service-Provider. Dort baute er einen eigenen Bereich zum Thema „Linux-Consultig" auf und konzipierte 1998 den ersten flächendeckenden Einsatz von Linux in einem deutschen Unternehmen. Alfred Schröder gründete 2001 die GONICUS GmbH. Hier ist er als geschäftsführender Gesellschafter für das operative Geschäft verantwortlich. Dabei gilt sein Interesse nach wie vor den Möglichkeiten des Einsatzes von GNU/Linux. Zugleich ist er Leiter des Arbeitskreises Desktop innerhalb des LIVE-Linux-Verbandes.

Thomas Sprickmann Kerkerinck begann nach dem Studium der Wirtschaftswissenschaften in Münster 1993 seine berufliche Laufbahn bei einem großen Dienstleistungsunternehmen in Berlin. Mit den während der Mitarbeit an verschiedenen Großprojekten erworbenen Qualifikationen wechselte er 1999 als Vertriebs-

leiter in ein Software- und Systemhaus in Dortmund, wo er unter anderem erste Erfahrungen mit Linux sammelte. Diese Erfahrungen und das Praxiswissen um IT-Themen ermutigten ihn, 2001 mit 4 weiteren Kollegen die Natural Computing GmbH zu gründen. Die Natural Computing GmbH entwickelt Lösungen zum Einsatz von Linux am Arbeitsplatz in heterogenen IT-Landschaften. Seit Sommer 2003 ist Herr Sprickmann Kerkerinck Mitglied im Vorstand des LIVE Linux Verband e.V.

Patrick Stewin studiert seit Oktober 2000 Informatik an der Technischen Universität Berlin. Die Schwerpunkte seines Studiums setzt er in den Fachgebieten *Intelligente Netze und Management Verteilter Systeme* (im Studiengebiet Betriebs- und Kommunikationssysteme) und *Informatik und Gesellschaft* (am Institut für Wirtschaftsinformatik). Neben seinem Studium arbeitet Patrick Stewin als „Werksstudent" in einem kleinen IT-Unternehmen. In diesem Unternehmen konnte er seit etwa eineinhalb Jahren Erfahrungen mit Open-Source-Software (z.B. mit Content-Management-Systemen und Web- bzw. Applikationsservern) sammeln. Er hat an der Erstellung des Kapitels „Grundlagen und Erfahrungen" mitgewirkt.

Kerstin Terhoeven, Dr., absolvierte das Studium der Betriebswirtschaftslehre mit den Schwerpunkten Rechnungslegung, Marketing und Wirtschaftsinformatik an der RWTH Aachen. Seit 1999 ist sie wissenschaftliche Mitarbeiterin der Monopolkommission in Bonn mit den Themenbereichen statistische Konzentrationserfassung, Finanzdienstleistungen und Internetökonomie. Sie hat Nutzungserfahrungen mit den Betriebssystemfamilien DOS, Windows, MacOS und Linux sowie zahlreicher Anwendungsprogramme und einiger Programmiersprachen.

Martin Unger ist Student der Medien-/Wirtschaftsinformatik an der TU-Berlin mit Schwerpunkt „Informatik und Gesellschaft." Im Rahmen des Projekts „Open-Source-Jahrbuch 2004" war er zuständig für die Auswahl der Autoren und die Redaktion von Kapitel 4 „Recht und Politik," sowie den Entwurf und Aufbau der Internetpräsenz des Projekts.

Karsten Weber, Dr. phil., ist Philosoph und lehrt zurzeit am Lehrstuhl für Philosophische Grundlagen Kulturwissenschaftlicher Analyse der Europa-Universität Viadrina in Frankfurt (Oder). Nach einer Ausbildung zum EDV-Kaufmann studierte er Philosophie, Informatik und Soziologie in Karlsruhe, promovierte 1999 über ein interdisziplinäres Thema zwischen Wissenschaftstheorie und analytischer Philosophie des Geistes und habilitierte sich zum Thema „Ethik der Informationstechnologie" an der Europa-Universität Viadrina. 1996–99 arbeitete er als Wissenschaftlicher Mitarbeiter am Studium Generale und beim Deutsch-Russischen Kolleg der Universität Karlsruhe.

Andreas Wiebe, Prof. Dr. iur., LLM, ist Leiter der Abteilung für Informationsrecht und Immaterialgüterrecht an der Wirtschaftsuniversität Wien. Nach seinem Studium der Rechtswissenschaften in Hannover und in den USA, wo er zusätzlich

zum juristischen Staatsexamen den Grad des Master of Law erwarb, promovierte er 1991 in Hannover zum Thema „Der wettbewerbsrechtliche Schutz von Computersoftware in Deutschland und den USA. Eine rechtsvergleichende Untersuchung unter besonderer Berücksichtigung des Know-how-Schutzes." Von 1990 bis 2001 war er als Akademischer Rat und wissenschaftlicher Assistent am Institut für Rechtsinformatik der Universität Hannover tätig. 2001 habilitierte er sich zum Thema „Die elektronische Willenserklärung – Rechtsgeschäftliche Grundlagen des elektronischen Geschäftsverkehrs." Seit 2002 ist er Professor für Privatrecht, insbesondere Informationsrecht und E-Commerce-Law an der Wirtschaftsuniversität Wien. Seine Schwerpunkte in Forschung und Lehre sind das EDV- und Informationsrecht sowie Gewerblicher Rechtsschutz und Urheberrecht. In den letzten Jahren stand dabei die Auseinandersetzung mit den Rechtsproblemen des Internet und des elektronischen Geschäftsverkehrs im Vordergrund.

Thomas Wieland, Dr., ist seit 2002 Professor für Telematik, Mobile Computing und Computergrafik an der Fachhochschule Coburg. Nach seiner Promotion an der Universität Bayreuth arbeitete er für das Deutsche Zentrum für Luft- und Raumfahrt (DLR) in Oberpfaffenhofen an großen Softwaresystemen für die Verarbeitung von Satellitendaten und anschließend mehrere Jahre für die Abteilung Corporate Technology der Siemens AG in München. Dort leitete er ein Forschungsteam im Bereich flexible Dienstvernetzung und Mobile Computing. Außerdem betreute er eine konzernweite Informationsplattform zu Linux und Open Source. Dr. Wieland war Gründungschefredakteur der Zeitschrift „Linux Enterprise" (2000–2001).

Uli Zappe studierte Philosophie, Soziologie, Psychoanalyse, Physik und Theaterwissenschaft. Er arbeitet an Projekten aus den Bereichen Philosophie, Musik, Film/Theater und Freie Software.

Thomas Zimmermann ist Sozialwissenschaftler und seit 2002 als wissenschaftlicher Mitarbeiter am Kolleg für Management und Gestaltung Nachhaltiger Entwicklung Berlin tätig und dort für die Bereiche E-Learning und IT-Projektmanagement zuständig. Er hat Studium an der Humboldt-Universität Berlin mit einer Diplomarbeit zum Thema „New Economy – veränderte sozio-ökonomische Rahmenbedingungen der Arbeit in der informationalen Wirtschaft" im Jahr 2002 abgeschlossen. Seine derzeitigen Arbeitsschwerpunkte sind „Informationale Ökonomie" sowie „Freie und offene sozio-ökoonomische Strukturen."