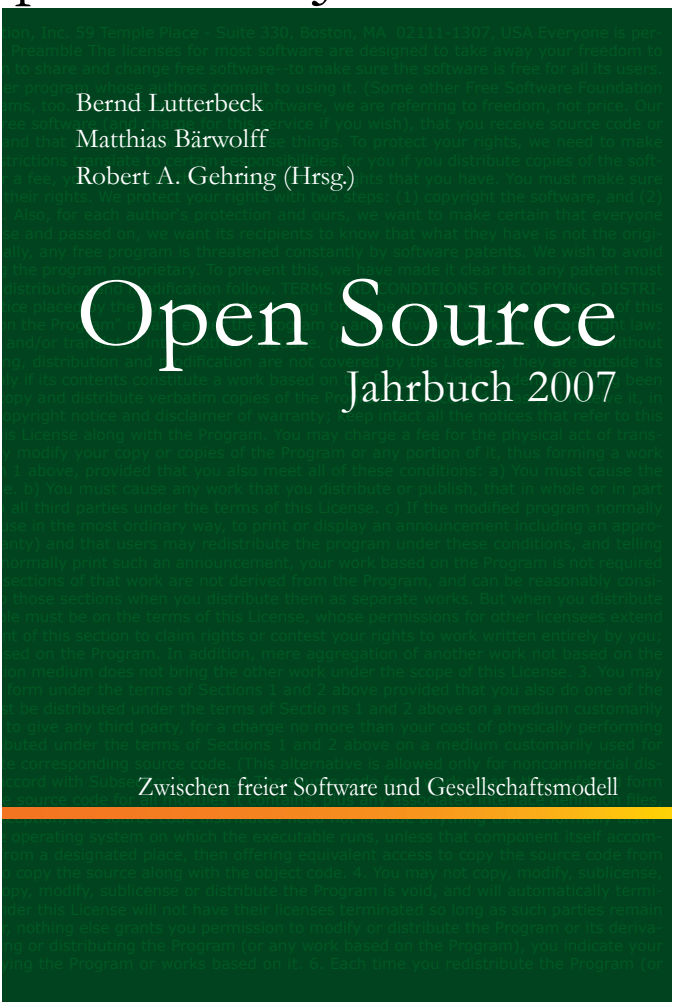
Dieser Artikel ist Teil des

Open Source Jahrbuchs 2007



erhältlich unter www.opensourcejahrbuch.de.

Die komplette Ausgabe enthält viele weitere interessante Artikel. Sie können diesen und andere Artikel im Open-Source-Jahrbuch-Portal kommentieren oder bewerten: www.opensourcejahrbuch.de/portal/. Lob und Kritik sowie weitere Anregungen können Sie uns auch per E-Mail mitteilen.

Open Source – ein aufstrebendes ökonomisches Modell*

Bruce Perens

(CC-Lizenz siehe Seite 563)

Die Entwickler von Open-Source-Software haben, vermutlich ohne bewusste Absicht, ein neues und erstaunlich erfolgreiches Modell der Softwareentwicklung geschaffen. Durch eine genauere Untersuchung dieses Modells können eine Reihe wichtiger Fragen beantwortet werden. Da nicht jedem sofort ersichtlich ist, wie Open Source auch wirtschaftlich funktionieren kann, wird diese Form der Softwareentwicklung leider häufig als auf lange Sicht unwirtschaftlich verkannt. Mancher befürchtet gar, dass die potenziellen negativen Auswirkungen auf die proprietäre Softwareindustrie der Gesamtwirtschaft Schaden zufügen können. Bei einer genaueren Betrachtung der allgemeinen wirtschaftlichen Funktionsweisen des Softwaresektors ist es jedoch ein Leichtes, zu zeigen, dass Open Source sowohl zukunftsfähig als auch von enormem Vorteil für die Gesamtwirtschaft ist. Konventionelle Theorien der freien Marktwirtschaft lassen sich eins zu eins auch auf Open Source anwenden. Dabei stellt sich heraus, dass Open Source viel enger mit dem Phänomen des Kapitalismus verbunden ist, als man vielleicht erwartet hätte.

Schlüsselwörter: Wirtschaftlichkeit · Produktdifferenzierung · ökonomische Modelle der Softwareentwicklung

1 Eine solide wirtschaftliche Grundlage

In den Anfängen der Open-Source-Bewegung haben auch Befürworter den wirtschaftlichen Aspekt des Modells nicht vollständig verstanden. Aufgrund dieses Mangels an Verständnis erzeugten wir den Eindruck, dass Open Source[1] keine solide wirtschaftliche Grundlage habe und trugen so zu der allgemeinen Ansicht bei, dass Open

* Aus dem Englischen von Katharina Mertens, Nadja Schüler, Angela Dirlick und Martin Görnitz.

1 Im Rahmen dieses Artikels können *Open Source* und *freie Software* als gleichbedeutend behandelt werden. Es existieren philosophische Differenzen zwischen den Vertretern beider Bewegungen, aber auf einen

Source weder zukunftsfähig sei, noch in der Lage, das Bedürfnis des Marktes nach neuer Technologie zu befriedigen. Dieser Eindruck bedarf inzwischen dringender Korrektur. Eric Raymond (1999) versuchte in seinem Werk „The Cathedral and the Bazaar“[2], Open Source als eine Art Geschenkökonomie zu erklären, ein Phänomen unter Computerprogrammierern, die Spaß daran haben, sich auf kreative Art und Weise einmal mit etwas zu beschäftigen, das nichts mit ihrer beruflichen Anstellung zu tun hat, und sich aus einem gestalterischen Anspruch heraus wünschen, dass ihre Arbeit wertgeschätzt wird. Raymond beschreibt dabei äußerst zutreffend das Verhalten von Programmierern in ihrer eigenen Subkultur: Die von ihm beschriebenen Beweggründe beherrschten tatsächlich zu Beginn die Open-Source-Szene und motivieren auch heute noch eine ganz entscheidende Gruppe der Mitwirkenden im Bereich Open Source.

Raymond suchte jedoch nicht nach Erklärungen dafür, warum sich große Unternehmen wie *IBM* an Open Source beteiligen, dazu kam es erst, nachdem sein Werk bereits erschienen war. Zum betreffenden Zeitpunkt hatte die Geschäftswelt gerade erst begonnen, sich ernsthaft für Open Source zu interessieren, und es handelte sich noch um kein bedeutsames wirtschaftliches Phänomen. „The Cathedral and the Bazaar“ kann daher nicht auf die Art von Einblicken in die wirtschaftliche Funktionsweise von Open Source zurückgreifen, die uns heute zur Verfügung stehen.

Diese ersten Ausführungen von Raymond wurden leider häufig fälschlicherweise als Beweis für das schwache ökonomische Fundament von Open Source interpretiert. Raymond zufolge wird der eigene Arbeitsaufwand eher mit immateriellen als mit finanziellen Gegenleistungen honoriert. Heute jedoch lässt sich erfreulicherweise feststellen, dass es für viele Entwickler von Open-Source-Software eben doch sehr hohe Verdienstmöglichkeiten gibt. Diese sind jedoch noch immer indirekter als in der Entwicklung proprietärer Software. Um die Ökonomie von Open Source zu verstehen, muss man daher tiefer vordringen, als dies bei der Betrachtung proprietärer Software notwendig wäre.

Großteil der Software, die per Definition Open Source ist, kann auch die Definition freier Software der *Free Software Foundation (FSF)* angewandt werden.

Es ist wichtig zu wissen, dass zu dem Zeitpunkt, an dem ich das Dokument, welches später als *Open Source Definition* bekannt wurde, als Richtlinie für das Debian-Projekt entwarf, die *FSF* ihre Definition freier Software noch nicht veröffentlicht hatte. Damals kommentierte Richard Stallman in einer persönlichen E-Mail, dass mein Dokument eine gute Definition freier Software darstellte.

Stallman widersprach jedoch meinem Argument, dass Unternehmen besser beraten wären, auf eine Free-Software-Lizenz zu verzichten, wenn dadurch der Umfang der Differenzierung für dieses Unternehmen geschmälert würde. Aus diesem Grund verwende ich im Folgenden die Bezeichnung Open Source, um Stallmans Ansichten bezüglich freier Software zu würdigen.

2 Raymond ergänzte sein Essay um zwei weitere Werke, „Homesteading the Noosphere“ (1998) und „The Magic Cauldron“ (1999), in denen er seine Betrachtung der Open-Source-Ökonomie fortsetzt. Alle drei Texte können unter http://www.catb.org/~esr/writings/ abgerufen werden.

2 Wie wird Ihr Unternehmen so erfolgreich wie Microsoft?

Angehenden Unternehmern der Softwarebranche wird häufig die Frage gestellt, wie sie es schaffen wollen, ebenso erfolgreich zu sein wie *Microsoft*. Wer sich mit seinem Unternehmen gar auf Open Source spezialisiert, wird zudem häufig vor die Frage gestellt, wie man mit freier Software an den Erfolg von *Microsoft* anschließen kann. Doch diese Frage ist schon an sich unberechtigt, wenn es doch das eigentliche Ziel ist, die Dinge besser zu machen als *Microsoft*. Sie spiegelt die Tatsache wider, dass bis zum heutigen Zeitpunkt viele Leute Software von einer sehr auf den Anbieter ausgerichteten Perspektive aus betrachten.

Fast jeder – ob er es zugibt oder nicht – lässt sich von der Finanzkraft und Überheblichkeit beeindrucken, die mit *Microsoft* assoziiert wird, und wenn es um die Herstellung von Software geht, denkt jeder gleich an *Microsoft* und deren Vorgehensweise. Es stellt sich jedoch heraus, dass das Microsoft-Modell nur einen Bruchteil der Software, die in der heutigen Geschäftswelt hergestellt und benutzt wird, für sich verbuchen kann. Nur etwa 30 Prozent der entwickelten Software wird auch als Software verkauft.[3] Ein großer Teil steht überhaupt nicht zum Verkauf. Er wird direkt für den Kunden entwickelt, von den eigenen Mitarbeitern oder externen Dienstleistern, die ihre Entwicklertätigkeit und nicht das Endprodukt in Rechnung stellen. Man muss sich vor Augen führen, warum dies so ist, um die ökonomischen Aspekte von Open Source nachvollziehen zu können.

3 Open Source als Reaktion auf eine allgemeine Unzufriedenheit

Im Februar 1998 gründeten Eric Raymond und ich die Open-Source-Initiative. Wir konnten auf exzellente Vorarbeit aufbauen: Die Free-Software-Kampagne, die auf Richard Stallman zurückgeht, gab es bereits seit 1983[4] und durch sie wurde ein großer Teil dessen geschaffen, was man heutzutage als Open Source bezeichnet. Die Verbindung des Linux-Betriebssystemkernels mit Stallmans GNU-System wurde nach

3 In einem Bericht des *U.S. Office of Technology and Electronic Commerce* (2003) zur Größe des US-Softwaremarkts repräsentiert *Packaged Software* 24,6 Prozent der Softwareindustrie. Der Rest des Entwicklungssektors besteht aus *Computer Programming Services*, *Computer Integrated Systems Design*, *Computer Processing and Data Preparation and Processing*, *Information Retrieval Services*, *Computer Facilities Management Services* sowie verschiedenen Untergruppen von *Software Publishing*.

Allerdings beinhaltet der Bericht keine Angaben zu Software, die für den firmeninternen Gebrauch durch angestellte Programmierer in einem Unternehmen, dessen Hauptprodukte nicht softwarebezogen sind, hergestellt wird. Zurückhaltend geschätzt würde ich die Zahl der internen Entwickler im Bereich der externen Auftragnehmer ansiedeln, die unter *Computer Programming Services* mit 19 Prozent Marktanteil angegeben werden.

4 Das GNU-Projekt wurde ursprünglich im September 1983 angekündigt (siehe http://www.gnu.org/gnu/initial-announcement.html), obwohl das Projekt erst im Januar 1984 begann. 1985 schließlich kam es dann zur Gründung der *Free Software Foundation*. Ein Überblick des GNU-Projekts findet sich unter http://www.gnu.org/gnu/gnu-history.html.

und nach auch für den Einsatz in Unternehmen brauchbar. Jedoch stellte Stallman Open Source auf eine Weise dar, welche die Akzeptanz der Werte gewisser Freiheiten *a priori* voraussetzte. Stallman ist Programmierer, und er entschied sich daher für eine philosophische Darstellung, die Programmierer ansprach.

Geschäftsleute hingegen sind Pragmatiker und lassen sich von wirtschaftlichen Vorteilen beeindrucken. Da die Art seiner Darstellung Stallmans Zielgruppe von vorneherein begrenzte, hatte seine Kampagne nicht die wirtschaftliche Schlagkraft, die sich heute abzeichnet.[5] Raymond und ich entschieden uns dafür, Geschäftsleute auf eine pragmatischere Art und Weise anzusprechen, in der Annahme, dass sie Stallmans Theorie eher wertschätzen würden, wenn sie einmal deren konkrete Vorteile erkannt hätten.[6]

Der Begriff Open Source kam der Öffentlichkeit zum ersten Mal in einer Bekanntmachung zu Ohren, die ich auf *Slashdot* und einigen Mailinglisten veröffentlichte. Sie beinhaltete eine Einführung und die Open-Source-Definition, die sowohl ein Open-Source-Grundsatzprogramm als auch eine Definition einer akzeptablen Vorgehensweise zur Open-Source-Software-Lizenzierung darstellte. Diese hatte ich sechs Monate zuvor als Grundsatzprogramm für das Debian-Projekt ausgearbeitet. Eric Raymond überarbeitete seinen Artikel „The Cathedral and the Bazaar", der ein Jahr zuvor erschienen war, und ersetzte darin die Worte *Free Software* durch *Open Source*.

Einige Zeit später fragte ein Reporter Steve Ballmer, den damaligen Präsidenten von *Microsoft*, ob man plane, *Windows* als Open-Source-Produkt zur Verfügung zu stellen. Ballmer erklärte daraufhin, dass Open Source nicht nur ein bestimmter Quellcode sei, sondern eine bestimmte Art der Software-Lizenzierung – der Präsident von *Microsoft* hatte mein Grundsatzprogramm gelesen und wurde von der Presse dazu befragt!

Veröffentlichungen von so unbekannten Menschen, wie Raymond und ich selbst es damals waren, erfahren diese Art von Aufmerksamkeit gemeinhin nur dann, wenn die

5 Hinzu kam, dass das GNU-Projekt eine Menge Software entwickelt und integriert hatte, es jedoch nicht geschafft hatte, einen eigenen Betriebssystemkernel zu entwickeln. Dies öffnete die Tür für Linus Torvalds, den letzten Teil des Projekts zu vollenden. Üblicherweise bezieht ein Betriebssystem seinen Namen vom Kernel, ungeachtet der Tatsache, dass es viele andere Komponenten abseits des Kernels enthält. Hier wiederfuhr Stallman ein Unrecht, das bis heute fortbesteht. Obwohl Stallman einen Großteil der Arbeit vollbrachte, die *Linux* erst möglich machte, fühlte sich Torvalds Team von Kernelentwicklern ihm nicht verpflichtet. In den Ankündigungen von *Linux* fand die *FSF* keine Erwähnung. In Wahrheit ist *Linux* jedoch nur der Kernel. Der Rest dessen, was üblicherweise in einer Linux-Distribution enthalten ist, besteht größtenteils aus den Komponenten von Stallmans *GNU System*. Stallman sind daher sowohl das GNU/Linux-Konzept, die Entwicklung des C-Compilers (der sicher in seiner Bedeutung dem Linux-Kernel ebenbürtig ist), der Emacs-Editor als auch die Anregungen, die er zur Entwicklung eines Großteils des übrigen Systems gab, zuzuschreiben.

6 In gewissem Umfang würdigte Raymond dabei Stallmans Ansichten herab, als er einen Konflikt mit Stallman und der *FSF* hätte vermeiden können, wenn er sich, der Einfachheit halber, nur an die Gruppe gewandt hätte, für die Open Source von Interesse ist und die sich von der, die Stallman anspricht, grundsätzlich unterscheidet. Meiner Ansicht nach beruht das bedauernswerte Resultat eher auf den gegensätzlichen Persönlichkeiten als auf unvereinbaren Philosophien.

Welt aus irgendeinem Grund auf sie gewartet hat. Ich behaupte, dass der Grund eine allgemeine Unzufriedenheit mit dem Marktführer im Bereich Softwareentwicklung war, nämlich *Microsoft* und seinen Produkten, das gemeinhin als ein Unternehmen galt, welches die eigenen Interessen über die des Kunden stellte. Es war bekannt, dass ihre Software unbehobene *Bugs* enthielt und dass ihr Desktop-Betriebssystem extrem anfällig für Systemabstürze war. Sie kamen damit „durch", da es keine nennenswerten Konkurrenten im Desktopbereich gab.

Die Menschen suchten nach einer Alternative. Wenn wir Software an sich verbessern wollten, mussten wir anders vorgehen als *Microsoft*. Open Source schaffte neue ökonomische Beziehungen. Eine neue Peer-to-Peer-Beziehung entstand zwischen Geschäftskunden von Softwareprodukten, die jetzt in der Lage waren, direkt an der Entwicklung von Open-Source-Software mitzuwirken und somit zu Softwareentwicklern füreinander wurden. Softwareanbieter entwickelten ein neues Verhältnis zueinander, da sie gemeinsam an Open-Source-Projekten arbeiteten, während sie in anderen Bereichen Konkurrenten waren. Und auch das Verhältnis von Kunden und Anbietern veränderte sich, da nun beide gemeinsam an der Entwicklung von Software beteiligt sein konnten. Aufgrund der Tatsache, dass viele Hersteller auf denselben Quelltext Zugriff hatten und somit das Produkt unterstützen konnten, verlor der *lock-in*, die unfreiwillige Gebundenheit an einen Hersteller, an Bedeutung, und die Beziehung zwischen Kunde und Anbieter war von weniger Exklusivität geprägt. Als Ergebnis all dieser Veränderungen fanden sich neue Anbieter auf dem Markt, und auch die Etablierten ersetzten nach und nach proprietäre Software durch Open-Source–Produkte.

Die gemeinschaftliche Beteiligung an der Erschaffung, dem Eigentum und den Vorteilen der Software stand in starkem Gegensatz zu dem Modell von *Microsoft*. Erste Anzeichen einer überraschenden Effizienz von Open Source waren die Schaffung eines technisch erfolgreichen Betriebssystems und der ersten praktisch anwendbaren Webserver und -clients. An diesem Punkt begann die Öffentlichkeit, sich massiv für Open Source zu interessieren und darin zu investieren. Es folgte ein enormes wirtschaftliches Wachstum: Der Linux-Sektor allein hat eine jährliche Wachstumsrate von 37 bis 45 Prozent, und bis zum Jahr 2008 erwartet man einen Marktumfang von 35 Milliarden US-Dollar (IDC Software Consulting 2004).

Diese ungewöhnliche Akzeptanz und die verblüffenden finanziellen Aussichten sprechen dafür, dass Open Source Ansprüche bedient, die vorher nicht in gewünschter Weise befriedigt wurden. Wäre dem nicht so, hätten sich keine so erfolgreichen und scheinbar absurden Phänomene entwickelt:

Linux, einst das Hobby eines Studenten Anfang 20, erobert den Enterprise-Computing-Markt.

IBM, der Inbegriff eines konservativen Unternehmens, zieht seinem Milliarden Dollar teuren AIX-Betriebssystem ein Produkt vor, das von einer zusammengewür-

felten Gruppe von Programmierern ohne gemeinsames finanzielles Interesse entwickelt wurde, die dem Unternehmen nicht weisungsgebunden sind, mit einem Vorsitzenden, dessen gesamter Einfluss auf dem Respekt der anderen beruht.

Microsoft hat seit einem Jahrzehnt den ersten ernsthaften Konkurrenten: Programmierer, welche die Produkte ihrer Arbeit verschenken.

All das erscheint zunächst keinen Sinn zu ergeben und den allgemeinen ökonomischen Grundregeln für den Bereich der Technologieentwicklung zu widersprechen. Es handelt sich hier um ein neues wirtschaftliches Phänomen, das sich nur erklären lässt, wenn man die wirtschaftlichen Gegebenheiten der Softwareentwicklung näher betrachtet.

4 Die Ursachen des wirtschaftlichen Erfolgs

Da *Microsoft* nach Ansicht der meisten Menschen als das bedeutsamste Modellunternehmen im Bereich der Softwareerzeugung gilt, und da wir uns hier mit den wirtschaftlichen Gegebenheiten der Softwarebranche befassen, sollten wir uns eine Frage stellen: Was macht die wirtschaftliche Stärke von *Microsoft* aus?

Man könnte zunächst vermuten, es sei der Wohlstand von Bill Gates, aber das ist natürlich Unsinn. Sein Reichtum wird von vielen geneidet, aber Bill Gates ist nicht gleichzusetzen mit *Microsoft*. Und auch die Tatsache, dass die Firma aktuell über einen liquiden Bestand von 70 Milliarden Dollar verfügt, somit das größte Softwareunternehmen der Welt ist und 71 000 Mitarbeiter in 103 Ländern beschäftigt, beantwortet unsere Frage nicht. Die Bedeutung von *Microsoft* ist vielmehr darauf zurückzuführen, dass viele Unternehmen, nämlich seine Kunden, durch Microsofts Software effizienter arbeiten können und sogar ohne die entsprechende Software überhaupt nicht mehr agieren könnten. *Microsoft* stellt Werkzeuge her, und sein wirtschaftlicher Einfluss ist marginal, verglichen mit der gemeinsamen Bedeutung all derer, die diese Werkzeuge verwenden. Der sekundäre ökonomische Einfluss all der Menschen und Unternehmen, die eine Technologie nutzen, ist größer als der primäre ökonomische Einfluss des Geldes, welches durch den Verkauf der Technologie eingenommen wird. Und all das gilt natürlich auch für Open-Source-Software.

5 Die wirtschaftliche Bedeutung von Software in Unternehmen

Wir müssen uns vor Augen führen, dass die meisten Unternehmen nicht im Bereich der Softwareherstellung tätig sind: Sie bieten Dienstleistungen an, verkaufen Schiffe, Schuhe oder andere Produkte. Dennoch sind alle Unternehmen bis auf die allerkleinsten in einem gewissen Maße von Software abhängig. Ohne sie wäre ihre Arbeit weniger effizient oder sogar überhaupt nicht durchführbar. Erinnern wir

uns einmal daran, wie Finanzplanung ohne Tabellenkalkulationsprogramme aussah, Geschäftskorrespondenz ohne E-Mails und an Kundeninterfaces bestehend aus Telefon mit Tonwahlverfahren als fortschrittlichstem Instrument der Kundeninteraktion. Schnell wird klar, dass im heutigen täglichen Geschäftsleben Software unabdingbar ist. Sie wird so dringend benötigt, dass jedes Unternehmen mit mehr als 50 Mitarbeitern, auch wenn es keine Software verkauft, einen Programmierer, einen Webdesigner oder einen Skript-programmierenden Systemadministrator beschäftigt.[7] Diese Unternehmen stellen keine Software her, aber dennoch ist Software für sie im täglichen Geschäftsleben von größter Wichtigkeit.

6 Gebrauchstechnologie im Vergleich zu Produktdifferenzierung

Gebrauchstechnologie ist unverzichtbar für ein Unternehmen, aber nicht das Produkt, das verkauft wird. Wer Bücher verkauft, für den sind Bücher das Ertragszentrum, während Software ein Kostenzentrum ist, eine unvermeidbare Ausgabe im Geschäftsleben. Die polemische Aussage, IT spiele keine Rolle, bedeutet im Grunde genommen, dass IT häufig kein Ertragszentrum ist, aber dennoch insofern benötigt wird, als dass sie Geschäfte überhaupt erst ermöglicht. IT-Technologie ist Gebrauchstechnologie.

Es gibt zwei grundlegende Arten kostenproduzierender Gebrauchstechnologie: differenzierende und nicht differenzierende. Differenzierende Technologie macht ein Unternehmen im Vergleich zu einem Konkurrenten für den Kunden ansprechender. Wenn man zum Beispiel die Website Amazon.com besucht, um ein bestimmtes Buch zu kaufen, dann bietet einem *Amazon* auch andere Bücher an, die von Kunden erworben wurden, die sich für dieses Buch entschieden haben. Die vorgeschlagenen Bücher sind tatsächlich häufig so interessant, dass man eines davon zusätzlich erwirbt. Auf der Seite von *Barnes & Noble* gibt es diese Funktion nicht, und es ist daher nicht verwunderlich, dass *Amazon* mehr Bücher online verkauft. Die Software, die diese Empfehlungen möglich macht, unterscheidet *Amazon* also von anderen Anbietern.[8] Offensichtlich wäre es ein Fehler, eine solche differenzierende Software als Open Source zur Verfügung zu stellen, da sie in diesem Fall von einem Konkurrenzunternehmen genutzt werden könnte, um genauso ansprechend für die Kunden zu werden.

---

7 Das *U.S. Bureau of Labor Statistics* führt 2 249 000 Stellen in programmierbezogenen Berufen in den USA: 455 000 Computerprogrammierer, 800 000 Softwareentwickler, 994 000 Systemanalysten, Datenbankadministratoren und Informatiker. Nicht enthalten sind die 779 000 Computer-Support–Spezialisten und Systemadministratoren, die zumindest ab und an Skriptprogrammierung betreiben, sowie Computeranwender, die 149 000 Positionen besetzen (U.S. Bureau of Labor Statistics 2006).

8 Ironischerweise ist dieses Feature Anlass einer Patentrechtsklage, die *Cendant* im November 2005 gegen *Amazon* eingereicht hat. Nach einer Erneuerung der Klage durch *Cendant* im Juni 2005 und einer Gegenklage von *Amazon* ist „Cendant vs. Amazon" bis heute noch nicht entschieden.

Auf der anderen Seite würde es einen Unternehmen nicht schaden, wenn der Konkurrenz die Funktionsweise ihrer nicht-differenzierenden Software bis ins kleinste Detail bekannt wäre. Die Konkurrenz könnte sich sogar als ihr bestmöglicher Partner herausstellen, wenn sich die Zusammenarbeit auf nicht-differenzierende Software beschränkt, einfach weil die Bedürfnisse der Konkurrenz den eigenen so ähnlich sind. Dieses Prinzip wird Tag für Tag in der Welt der Open Source bestätigt. *HP* und *IBM* entwickeln gemeinsam Software, die beiden Anbietern dabei hilft, ihre Systeme zu verkaufen, während sie in anderen, spezielleren Bereichen der Softwareentwicklung, in denen eine Differenzierung möglich und sinnvoll ist, erbitterte Konkurrenten bleiben.

Bis zu 90 Prozent der Software eines jeden Unternehmens ist nicht-differenzierend. Ein großer Teil davon gehört zur Infrastruktur, der Grundlage, auf der differenzierende Technologie aufbaut. Unter Infrastruktur versteht man Dinge wie Betriebssysteme, Webserver, Datenbanken, Java-Application-Server und andere Middleware, Graphical-User-Interface-Desktops und die Tools, die allgemein bei GUI-Desktops benutzt werden, wie beispielsweise Webbrowser, E-Mail-Clients, Tabellenkalkulationsprogramme, Textverarbeitung und Präsentationsanwendungen. Jede Software, die für ein Nicht-Softwareunternehmen differenzierende Effekte hat, setzt auf einer oder mehrerer solcher Infrastrukturkomponenten auf. Ein wichtiger Indikator dafür, ob Software differenzierend ist oder nicht, ist die Frage, ob einem Konkurrenten dieselbe Software zur Verfügung steht. Weder Microsoft-Software, noch *Linux* und Open Source können auf lange Sicht dabei helfen, ein Unternehmen von anderen abzuheben, da sie jedem zur Verfügung stehen. Sie unterscheiden sich voneinander, können aber ihr Unternehmen nicht von anderen differenzieren. Durch die Wahl des einen oder anderen können Sie vielleicht Geld sparen oder die eigene Effizienz erhöhen, aber in der Regel nicht attraktiver für Ihre Kunden werden.

Ein weiterer wichtiger Indikator dafür, ob eine Software differenzierend ist oder nicht, ist die Frage, ob ihre Auswirkungen für den Kunden ersichtlich sind. Für Kunden ist es unbedeutend, welches Betriebssystem verwendet wird, solange die Systeme nicht ständig abstürzen. Für die Kunden ist es unbedeutend, ob *Microsoft Office* oder *OpenOffice.org* verwendet wird. Die Gründe für das eine oder das andere System mögen für Ihre Firma bedeutsam sein, für den Kunden aber sind sie irrelevant.

Um das eigene Unternehmen ansprechender für potenzielle Kunden zu gestalten, sollte man daher mehr Ressourcen in differenzierende Software investieren, die dieses Ziel unterstützt, und weniger in nicht-differenzierende. Open Source ist der Schlüssel dazu, weniger für Letztere auszugeben, indem Kosten und Risiken, die zuvor nur Ihr Unternehmen allein tragen musste, jetzt auf verschiedene, zusammenarbeitende Unternehmen aufgeteilt werden.

Dabei muss natürlich ehrlich eingeschätzt werden, welche Software tatsächlich differenzierend ist und welche nicht, was sich als schwierig herausstellt. In vielen Firmen sind Manager und Techniker nicht bereit, der Arbeit von Außenstehenden

Beachtung zu schenken, weil sie nicht glauben, dass diese genauso gut sein könnte wie die eigene. Dieses Phänomen, auch als *Not-Invented-Here-Syndrom* (NIH) bekannt, kommt einem teuer zu stehen: Ihre Mitarbeiter duplizieren die Arbeit, die andere schon geleistet haben, anstatt ihre Zeit mit der für den Betrieb so wichtigen differenzierenden Software zu verbringen.

Die Beteiligung an Open Source erhöht die Effizienz von Software-Cost-Centern, da Kosten und Risiken geteilt werden, die sie sonst alleine tragen müssten. Das Geld, welches so gespart wird, kann entweder die Gewinne erhöhen oder in andere Bereiche, wo es dringender benötigt wird, investiert werden.ihre

7 Modelle der Softwareentwicklung

Da Unternehmen nun einmal Software benötigen, muss diese auf die eine oder andere Art entwickelt werden. Zu den wichtigsten Methoden der Softwareentwicklung gehören:

- Entwicklung für den Vertrieb
- firmeninterne und Auftragsentwicklung
- Kollaboration ohne Open-Source-Lizenzierung
- Open Source

Diese Methoden unterscheiden sich in:

- ihrer Kostenverteilung
- ihrer Risikoverteilung
- ihrer Effizienz, mit der Geld in die Softwareentwicklung fließt und nicht in Gemeinkosten
- der Möglichkeit, andere von der Nutzung der Software auszuschließen

Diese Faktoren bestimmen, welche Methode für die Entwicklung einer bestimmten Software brauchbar ist. Dabei darf nicht vergessen werden, dass letztendlich der Kunde die Software finanziert, nicht der Anbieter. Der Kunde kann sein Geld in Projekten verwenden, welche eine beliebige dieser Methoden nutzen, um die benötigte Software zu beschaffen.

7.1 Entwicklung für den Vertrieb

Mit diesem Modell sind wir am vertrautesten, und das, obwohl weniger als 30 Prozent aller Software für den Vertrieb entwickelt wird.[9] Hier werden die Kosten der

9 Vgl. die Ausführungen zu *Packaged Software* unter obiger Fußnote 3.

Softwareentwicklung in der Regel von einem Hersteller allein getragen. Dieser wird versuchen, durch den Verkauf des fertigen Produkts sowohl seine Kosten zu amortisieren, als auch einen Profit zu erzielen. Dabei konzentrieren sich die Ausgaben in der Entwicklungsphase, und erst mit der Fertigstellung des Produkts kann der Hersteller anfangen, seine Kosten auszugleichen. Das Risiko eines nicht profitablen Produkts wird dabei allein vom Hersteller getragen.

Wenn sich das Produkt am Markt als erfolgreich erweist, werden die Kosten der Entwicklung nach und nach auf die Kunden verteilt. Da bei diesem Modell Kosten und Risiken nicht verteilt werden können, bis das Produkt fertig gestellt ist, benötigt der Hersteller häufig die Hilfe des Anlagemarkts, um Kosten und Risiken extern zu verteilen. Externes Kapital wird über einen langen Zeitraum hinweg benötigt, da die Entwicklung der Software und des dazugehörigen Marktes Jahre dauern kann und häufig noch mehr Zeit vergeht, bevor sich die Entwicklungskosten amortisieren und ein Profit möglich wird. Aktienmärkte erhöhen die Liquidität des Investments, da hier Investoren die Möglichkeit haben, die Einschätzung der Zukunftsaussichten einer Firma, die sich im Aktienwert widerspiegelt und nicht die tatsächlichen Umsätze des Unternehmens, in Geld zu verwandeln. Erfolgreiche Unternehmen können in die Entwicklung neuer Softwareprodukte reinvestieren, anstatt sich wieder dem Kapitalmarkt anzuvertrauen.

Die Gemeinkosten des traditionellen, nicht virtuellen Vertriebs sind extrem hoch, was dazu führt, dass weniger als 10 Prozent des Geldes, welches der Endverbraucher für ein Produkt zahlt, tatsächlich in die Vermarktung des Produkts, die Entwicklung und die Dokumentation fließen. *Microsoft* hat von seinen Verkaufseinnahmen aus dem Jahr 2005 gerade einmal 15 Prozent in die Forschung und die Entwicklung neuer Produkte investiert. Der restliche Umsatz wurde für Posten genutzt, von denen der Kunde nicht direkt profitiert, wie die sehr kostenintensive Kundenakquise für Microsoft-Produkte: Ausgaben für Werbung, Gestaltung und Herstellung attraktiver Verpackungen, die direkt nach dem Kauf entsorgt werden, Zahlungen an Einzelhändler für die Stellfläche auf dem Regal, in dem das Produkt ausgestellt wird, Verkaufspersonal und Gewinn. Die angegebenen 15 Prozent beinhalten dabei noch nicht den Preisaufschlag durch Einzel- und Großhändler; rechnet man diesen mit ein, dann sinkt der Anteil des Produktpreises, der durch die Entwicklung der Software gerechtfertigt ist, auf weit unter 10 Prozent (Microsoft Corp. 2006).

Auch auf der Erwerbsseite kommt es zu Ineffizienzen, da die gekaufte Software oft nicht dem entspricht, was der Kunde wirklich benötigt. Häufig wird Software gekauft, die sich bei genauerer Betrachtung als nicht brauchbar für die entsprechende Anwendung erweist oder nicht eingesetzt wird. Die Kosten für solche *Staubfänger* kann der Kunde häufig nicht ersetzen. Verluste kommen auch dadurch zustande, dass Softwareunternehmen Produkte aufgeben oder auslaufen lassen und nicht mehr unterstützen, die sich beim Kunden noch nicht amortisiert haben. Da selbst für Produkte, die ausgelaufen sind, ihr Quellcode nicht zur Verfügung gestellt wird, können

sie nicht weiter unterstützt werden, und dem Kunden bleibt nichts anderes übrig, als seine Investition abzuschreiben. Wenn wir all diese Risikofaktoren zusammenrechnen, dann werden, vorsichtig geschätzt, 50 Prozent (Krass 2002) der über den Handel erworbenen Software nicht genutzt oder nicht vollständig und effizient angewendet, so dass die Anschaffung ein Fehlschlag war.

Wenn wir zu dieser 50 Prozent Ausfallquote die Effizienz von weniger als 10 Prozent, mit der bei Vertriebsentwicklungen Geld in die Entwicklung reinvestiert wird, hinzurechnen, dann ergibt sich, dass gerade 5 Prozent des Geldes, welches für den Erwerb von Software ausgegeben wird, wieder in die Produktentwicklung fließt.

Das wichtigste Ergebnis dieser extrem niedrigen Effizienz ist, dass das Vertriebsmodell nur für die Entwicklung von Massenprodukten ökonomisch sinnvoll ist. Ein Massenmarkt verschleiert den Mangel an Effizienz, da jeder Kunde im Vergleich zu den Kosten für die Softwareentwicklung nur einen relativ geringen Betrag ausgeben muss. In der Gesamtsumme ergibt sich jedoch ein Betrag, der mehr als zwanzig mal so hoch ist wie die Entwicklungskosten der Software. Viele wichtige Softwareprodukte könnten für den Vertrieb überhaupt nicht geschaffen werden, da ihr Markt nicht groß genug wäre, um sowohl die Kosten für die Entwicklung als auch die hohen Gemeinkosten wieder auszugleichen. Außerdem wird ein Großteil der Produkte, obwohl sie potenziell viele Käufer finden könnten, von den Herstellern abgelehnt, weil es nicht gelingt, Unternehmen und Investoren davon zu überzeugen, dass sich ein entsprechender Markt entwickelt oder weil das Risiko zu hoch ist. Die Innovationskraft wird gedämpft, wenn ein Softwarehersteller seine Produkte über den Handel vertreibt.

Ein aufschlussreiches Beispiel für die Unfähigkeit zur Innovation ist die Tatsache, dass die wichtigsten Neuentwicklungen der weltweiten Computerbranche in den letzten zehn Jahren, Webserver und Browser, als Open-Source-Produkte im Rahmen eines von der Regierung finanzierten Universitätsprojekts entwickelt werden mussten. Keine der Firmen, denen die Fertigstellung des Produkts möglich gewesen wäre, ließ sich davon überzeugen, dass es profitabel wäre. Das einzige Unternehmen, das größere Investitionen getätigt hat, um das Internet zu entwickeln (*Autodesk*, mit der Investition in Ted Nelsons Xanadu-Projekt), entschied sich gegen die Fertigstellung des Produkts, da das Ertragsmodell, das sich *Xanadu* für das Internet vorstellte (Zahlungen an die Provider von Inhalten für jedes Wort und eine komplizierte Zuordnung von Derivaten und Referenzen) zu umständlich war. Tim Berners-Lee, letztendlich der führende Kopf bei der Entwicklung eines erfolgreichen Internets, hielt es für nicht notwendig, ein normiertes Modell zur Generierung von Einnahmen zu entwerfen und überließ es stattdessen den Nutzern, wie sie aus dem Internet Gewinne schlagen.

Wie schon zuvor erwähnt, ist Software, die wir im Geschäft erwerben, in der Regel jedem zugänglich, da sie an so viele Kunden wie möglich verkauft werden muss, um den höchstmöglichen Profit zu erreichen. Deshalb ist es schwierig oder sogar unmöglich, mit Hilfe von Software, die dem Vertriebsmodell entstammt, das Unternehmen des Kunden von anderen zu differenzieren.

7.2 Firmeninterne und Auftragsentwicklung

Während firmeninterne Entwicklungen von Programmierern durchgeführt werden, die direkt beim Kunden angestellt sind, wird eine Auftragsentwicklung von einer externen Firma übernommen, wobei der Auftraggeber exklusive Nutzrechte an dem Endprodukt erhält. In beiden Fällen werden die Programmierer üblicherweise für das Schreiben der Software bezahlt, anstatt für das fertige Produkt, wie es beim zuvor beschriebenen Vertriebsmodell der Fall wäre.

Eine andere Form der Auftragsentwicklung ist eine umfangreiche Anpassung durch den Verkäufer eines sonst frei verkäuflichen Produkts an die Wünsche eines speziellen Kunden. So könnte beispielsweise ein mittleres oder großes Unternehmen eine Standard-Webserver-Umgebung erwerben und dann den Verkäufer dafür bezahlen, das Produkt an seine speziellen Bedürfnisse anzupassen.

Sowohl im Fall firmeninterner Entwicklung als auch bei der Auftragsarbeit besitzt der Kunde die volle Kontrolle, da die Programmierer nur dann bezahlt werden, wenn sie seinen Wünschen gerecht geworden sind. In der Regel kann der Kunde auch bestimmen, ob er selbst das alleinige Nutzungsrecht für die Software erhält oder ob sie auch anderen zugänglich gemacht wird. Aufgrund dieser Kontrolle des Vertriebs durch den Kunden handelt es sich um ein ideales Modell zur Entwicklung von differenzierender Software, wenn nicht sogar um das einzige, mittels welchem Software entwickelt werden kann, die das Unternehmen des Endnutzers von anderen Unternehmen differenziert.

Es ist üblich, dass ein Auftragnehmer versucht, einen Teil der Arbeit, die er für einen bestimmten Auftraggeber durchgeführt hat, dafür zu nutzen, seine Geschäfte durch den Verkauf bereits entwickelter Produkte an andere Kunden in Schwung zu bringen. Wie erfolgreich solche Unternehmungen sind, hängt von den genauen Gegebenheiten des Vertrags und der Ehrlichkeit des Auftragnehmers ab. Wenn dieser erwarten kann, seine Arbeit auch an andere Kunden verkaufen zu können, berechnet er vielleicht einen geringeren Betrag und ein Teil der Kosten und Risiken können auf den Auftragnehmer und seinen nächsten Kunden übertragen werden. Im Falle von differenzierender Software ist dies jedoch eher ungünstig: Verkauft der Auftragnehmer die differenzierenden Merkmale eines Kunden weiter an andere Firmen, kann es schnell zu Problemen kommen.

Um in den Genuss der vollen Kontrolle zu gelangen, die mit Vertrags- oder firmeninterner Entwicklung verbunden ist, trägt der Kunde die Kosten und Risiken der Entwicklung meist allein. Für die Entwicklung von nicht-differenzierender Software gibt es daher oft kostengünstigere Möglichkeiten. Wer die vollständigen Kosten für die Entwicklung einer Software auf sich nimmt, die nur die Funktionsweise einer bereits existierenden Software kopiert, erfindet sozusagen das Rad neu und verschwendet damit Geld. Wer nicht auf die alleinige Nutzung eines Produkts und die vollständige Kontrolle über seine Entwicklung besteht, für den mag ein Open-Source-Produkt oder für den Vertrieb entwickelte Software kosteneffektiver sein.

Bei interner und Auftragsentwicklung fließt der größte Teil des investierten Geldes tatsächlich in die Entwicklung, nämlich 50–80 Prozent, verglichen mit den 10 Prozent im Falle des Verkaufs über den Handel. Die Abschläge entstehen hier durch die Kosten für die Kundenakquise des Auftragnehmers, die Kosten der Bereitstellung nicht dauerhaft genutzter Fachkompetenzen und durch die Gewinnmargen des Auftragnehmers.

Häufig verfehlen firmeninterne und Auftragsentwicklung das Ziel einer funktionierenden Software, die im Unternehmen des Kunden eingesetzt werden kann und die geforderten Ansprüche erfüllt. Diesen Entwicklungsmodellen kann eine Erfolgsrate von 50 Prozent (U.S. General Accountion Office 1997), ähnlich dem Vertriebsmodell, bescheinigt werden.

7.3 Kollaboration ohne Open-Source-Lizenzierung

Firmenübergreifende Zusammenarbeit in der Softwareentwicklung fand in der Vergangenheit üblicherweise im Rahmen von Konsortien statt, wobei es nicht selten zu gigantischen Fehlschlägen kam. In jüngerer Zeit wurden milliardenschwere geschlossene Konsortien häufig durch erfolgreichere Open-Source-Projekte ersetzt. Man erinnere sich an *Taligent* und *Monterey*, zwei Konsortien, die zu dem Zweck gebildet wurden, einen UNIX-Ersatz zu schaffen. *Linux* ersetzte sie. Oder das Common-Desktop–Environment-Projekt, welches in den meisten Firmen, die vorher *CDE* unterstützten, durch den Open-Source-Desktop *GNOME* ersetzt wurde.

Vor einigen Jahren gründete das große Landwirtschaftsunternehmen *Cargill* ein Konsortium in Form einer so genannten *gated community*: Ziel war es, die eigenen Partner in den Genuss der Vorteile von Open Source kommen zu lassen, jedoch gleichzeitig für Geheimhaltung zu sorgen und die Vorteile der Software nur mit den Mitgliedern des Konsortiums zu teilen. Zwei Jahre später verabschiedete *Cargill* sich von dem Projekt. Nicht-differenzierende Software in einem geschlossenen Konsortium zu entwickeln, ist einfach nicht angemessen. Wo es nicht um den Schutz differenzierender Merkmale geht, ist es sinnvoll, die Türen weit zu öffnen und Mitglieder zuzulassen, die nützliche Beiträge leisten können, auch wenn sie sich finanziell nicht beteiligen können. Ein Konsortium ist kostenintensiver, da weniger Mitglieder die Kosten und Risiken unter sich aufteilen als bei einem Open-Source-Projekt und das im Rahmen eines aufwändigeren Gefüges mit höheren Gemeinkosten, als es in einem entsprechenden Open-Source-Projekt notwendig wäre. Geschlossene Konsortien vertreten im Allgemeinen das Prinzip *pay-for-say*, d. h. nur wer sich finanziell beteiligt, darf auch mitentscheiden; Open Source dagegen setzt auf fachliche Leistung. Im Falle von *pay-for-say* kann ein Mitglied auf eine Schädigung des gesamten Projekts hinarbeiten, wenn dies zu seinem eigenen Vorteil ist. Produktplanung in Konsortien führt häufig zu unlösbaren Differenzen zwischen den beteiligten Firmen, da diese unterschiedliche Vorstellungen vom Vertrieb eines Produktes haben und Unstimmigkeiten zwischen Unternehmen im Marketing subjektiv und schwer zu lösen sind.

Betrachtet man die zahlreichen Fehlschläge von Konsortien und auf der anderen Seite die hohe Erfolgsquote großer Open-Source-Projekte zwischen denselben Unternehmen, lässt sich schließen, dass die Fairness, die mit Open-Source-Lizenzerteilung einhergeht, ein entscheidender Bestandteil effektiver Kollaboration zwischen einer großen Zahl von Teilnehmern mit unterschiedlichen Interessen ist.

7.4 Open Source

Das Open-Source-Modell beinhaltet die Zusammenarbeit zahlreicher Beteiligter (Individuen, Unternehmen, akademische Institutionen und andere) bei der Entwicklung eines Softwareprodukts.

Normalerweise wird die initiale Entwicklung von einer einzelnen Partei durchgeführt, wie es auch bei der innerbetrieblichen Entwicklung oder der Auftragsentwicklung der Fall ist. Die Software wird der Öffentlichkeit verfügbar gemacht, sobald sie für andere nutzbringend ist, aber im Allgemeinen vor der endgültigen Fertigstellung, und damit lange bevor ein Produkt im Handel erhältlich wäre. Sobald die Software nutzbar ist, machen andere Parteien sie sich zunutze. Und auch erst in diesem Moment, wenn das Produkt anderen einen Nutzen bietet, funktioniert das Open-Source-Modell wirklich, da erst jetzt auch eine Motivation besteht, die Software um zusätzliche Eigenschaften zu ergänzen, die für den Einzelnen interessant sind. Diese Erweiterung geschieht durch die Angestellten eines Kunden oder durch Auftragnehmer in dessen Kontrollbereich.

Die inkrementellen Kosten für das Hinzufügen eines Features sind dabei viel geringer als die Kosten für eine vollständige Entwicklung. Parteien, die Modifizierungen vornehmen, haben ein Interesse daran, diese so zu gestalten, dass sie von den anderen Entwicklern, die an dem Projekt beteiligt sind, akzeptiert werden und so in den Kern des Quellcodes eingearbeitet werden, den alle Entwickler teilen. Erfolgt diese Einbindung nicht, ist es kostenintensiver, die hinzugefügten Features aufrechtzuerhalten, da es Aufgabe der entwickelnden Partei bleibt, Änderungen am Hauptquellcode zu verfolgen und die Kompatibilität mit der sich verändernden *Codebase*, der Gesamtheit des Quellcodes, aufrechtzuerhalten.

Open Source führt daher häufig zur Bildung einer Gemeinschaft von Entwicklern, welche dazu beitragen, ein nützliches Produkt zu schaffen. Die Kosten und Risiken der Entwicklung werden unter den Programmierern aufgeteilt, und sollte jemand das Projekt verlassen, so können es die Übrigen fortführen. Die Aufteilung von Kosten und Risiken beginnt in dem Moment, an dem das Projekt reif genug ist, dass sich eine Community außerhalb des initialen Entwicklerkreises bildet.

Open-Source-Software wird direkt von ihren Endnutzern entwickelt. So werden zum Beispiel Features des Apache-Webservers oft von den Unternehmen hinzugefügt, die diese für ihre eigenen Websites benötigen, oder das Untermehmen beauftragt Dritte, die entsprechenden Erweiterungen vorzunehmen.

Die Kunden eines bestimmten Open-Source-Produktes geben sich in der Regel zu erkennen: Sie suchen in einem Open-Source-Software-Verzeichnis nach dem Produkt, laden es herunter und testen die Software. Wenn diese Tests erfolgreich verlaufen, setzen sie die Software für sich ein. Sie entwickeln somit ein fortdauerndes Interesse an dem Produkt. Wenn sie in dieser Situation eine Erweiterung benötigen, besteht ein Anreiz, sich an der weiteren Entwicklung der genutzten Software zu beteiligen.

Unternehmen, die sich an Open-Source-Kollaborationen beteiligen, möchten die jeweilige Software nicht-differenzierend und im Sinne eines Kostenzentrums nutzen. Für sie ist es nicht von Bedeutung, dass Open Source an sich keinen Profit schafft. Durch die Software sind sie in der Lage, in anderen Bereichen Profite zu machen, und damit dies so bleibt, müssen sie in ihre Kostenzentren investieren. Im Falle von differenzierender Software bleibt ihnen wenig anderes übrig, als das Produkt in Form einer firmeninternen oder einer Auftragsentwicklung zu erstellen, damit die differenzierenden Merkmale nicht von Konkurrenten genutzt werden können. Dagegen haben sie im Falle von nicht-differenzierender Software die Wahl zwischen einem Vertriebsprodukt und Open Source. Es stellt sich die Frage, was wirtschaftlicher ist.

Da sich die Kunden zu erkennen geben, leidet Open Source nicht an der Ineffizienz einer Vertriebsentwicklung, welche Werbung oder andere kostenintensive Mechanismen zur Kundengewinnung mit sich bringt. Auch ist Open Source wenigstens genauso effizient, wie es eine Eigen- oder Auftragsentwicklung wäre, wenn es darum geht, das zur Verfügung stehende Geld in die Softwareentwicklung anstatt in Gemeinkosten fließen zu lassen.

Aufgrund der höheren Effizienz in diesem Bereich können mit Hilfe von Open Source selbst Produkte entwickelt werden, die keinen Massenmarkt ansprechen und daher nicht wirtschaftlich erfolgreich im Rahmen des Vertriebsmodells entwickelt werden könnten.

Craig Small, Ingenieur und Entwickler von Netzwerk-Infrastrukturen, fasst die Vorteile für seine Kunden wie folgt zusammen:

> „Durch die Nutzung von Open Source haben sich die Initialkosten für Software drastisch reduzieren lassen. Dies war deshalb so wichtig, da sich so das Interesse dieses bestimmten Kunden wecken ließ. Ich hatte geplant, das Managementsystem des Netzwerks von Grund auf neu zu entwickeln, als ich auf ein Projekt stieß, das nahe genug an dem dran war, was ich benötigte. Ich war in der Lage, für das Problem nur einen Bruchteil der Arbeitsstunden aufzuwenden, die ich benötigt hätte, wenn ich von Null hätte anfangen müssen.
>
> Die Erweiterung und Anpassung von Open-Source-Software ist wesentlich einfacher, da ihre Entwickler von vorneherein davon ausgehen, dass andere ihre Arbeit auf eine Art ausbauen werden, die sie selbst nicht eingeplant hatten. Daher schreiben sie so, dass Ergänzungen einfacher

einzubinden sind. Die Programmierer proprietärer Software schreiben, als ob niemand außerhalb der eigenen Firma jemals ihren Quellcode lesen wird. Wir begannen mit einem Open-Source-Projekt, das für eine Hardware gedacht war, die völlig anders war als unsere eigene, konnten es aber leicht so abändern, dass sich unsere Anlagen überwachen ließen. Ich habe meine Erweiterungen wiederum dem Open-Source-Projekt zur Verfügung gestellt. So kann ich darauf vertrauen, dass für den Fall, dass ich die Firma verlasse, sich dennoch jemand, zumindest ein Teilnehmer des Projekts, weiterhin meiner Erweiterungen annimmt.

Inzwischen existiert eine ganze Gemeinschaft von Menschen, sowohl Entwicklern als auch nicht-programmierenden Nutzern, mit denselben technischen Bedürfnissen, die sich um das Softwareprojekt, zu dem wir alle beigetragen haben, entwickelt hat. Wir befinden uns im Austausch darüber, wie wir unsere Bedürfnisse besser erfüllen können und tragen so Ideen zusammen, auf die wir alleine nie gekommen wären.

Proprietäre Software birgt Kosten in sich, über die Unternehmen erst seit kurzer Zeit nachdenken. Organisationen wie die *Business Software Alliance* haben mit der Bundespolizei Razzien durchgeführt, bei denen sie von Unternehmen schriftliche Nachweise darüber forderten, dass sie für jede einzelne Software auf ihrem Gelände eine Lizenz erworben hatten. Sehr viel Zeit und Mühe wird dafür aufgebracht, sicherzustellen, dass sich Unternehmen an immer lästigere Lizenzbestimmungen halten. Warum ist es mir rechtlich gesehen nicht möglich, mein *Windows XP* von meinem Laptop auf meinen Desktop zu übertragen? Durch Open Source wird solch unproduktiver Nonsens vermieden."

Nicht alle Open-Source-Projekte sind erfolgreich. Viele scheitern bereits früh oder bleiben Solo-Projekte, die nur geringe Ressourcen in Anspruch nehmen. Die Ausgaben für solche Projekte sind jedoch unbedeutend, da sie eingestellt werden, bevor sie erste bedeutende Kunden oder eine Gemeinde von Programmierern anziehen. Reifere Open-Source-Projekte können eingestellt werden, wenn sich eine bessere Alternative ergibt, aber dabei ist es häufig möglich, den Quellcodes, die Daten und Erkenntnisse eines Projekts für ein anderes zu verwenden, und so gehen die Investitionen in die Entwicklung des Projekts nicht verloren.

Die Kosten der Teilnahme an fortgeschrittenen Open-Source-Projekten unterscheiden sich stark von den Kosten einer Vertriebsentwicklung bzw. interner oder Auftragsentwicklung. Der größte Teil der Kosten entsteht aufgrund des Zeitaufwands, den Mitarbeiter auf die Teilnahme am Projekt verwenden. Dieser Posten setzt sich zusammen aus den Personalkosten der Softwareevaluation, den Kosten für eigenes Personal oder Auftragnehmer, die eine existierende Open-Source-Software den Kundenbedürfnissen anpassen, und den Kosten für den Support der Software für interne Nutzer. Außerdem besteht immer die Möglichkeit, dass Zeit in eine Software inves-

tiert wird, welche dann doch ersetzt wird, da sie die jeweiligen Bedürfnisse doch nicht den Erwartungen entsprechend erfüllen kann.[10] Die maximalen Kosten für Open Source würden sich ergeben, wenn keine Gemeinschaft außerhalb des Kunden selbst bestünde: Die Kosten wären dann vergleichbar mit denen einer Auftragsentwicklung oder einer betriebsinternen Entwicklung, bei welcher der einzelne Kunde die Gesamtkosten trägt. Die tatsächlichen Kosten sind geringer, je nachdem wieviele Teilnehmer aktiv mitarbeiten und je nachdem wieviel Arbeit notwendig ist. Die Investition, die man verliert, liegt in der Hauptsache in den aufgewandten Personalkosten. Das mit eingerechnet, ergibt sich durch Open Source eine wirtschaftliche Effizienz, die mindestens so hoch ist, wie die einer firmeninternen oder Auftragsentwicklung und viel höher als die einer Entwicklung zu Vertriebszwecken.

7.5 Zusammenfassung der Methoden der Softwareentwicklung

Open Source hat mehrere entscheidende ökonomische Vorteile gegenüber allen anderen Methoden der Softwareentwicklung: Hier verbinden sich Effizienz in der Zuweisung von Ressourcen mit einer besseren Kosten- und Risikenverteilung als bei jeder anderen gängigen Methode der Softwareentwicklung. So können Produkte entwickelt werden, die im Vertrieb nicht kosteneffizient vertrieben werden könnten, weil es für sie keinen genügend großen Markt gibt. Bei Open Source können die Kosten und Risiken von Projekten verteilt werden, während bei einer Entwicklung durch die eigene Firma oder einen Auftragnehmer eine Partei alle Kosten und Risiken tragen muss. Open Source verteilt Kosten und Risiken direkt, ohne sich an den Anlagemarkt wenden zu müssen. Die Verteilung von Kosten und Risiken beginnt viel früher als es beim Vertriebsmodell der Fall wäre. Open Source gibt dem Kunden die volle Kontrolle über die Anpassung des Produktes. Der Kunde hat jedoch keine Kontrolle darüber, wer Zugang zu dem Produkt hat. Aus diesem Grund ist Open Source in den meisten Fällen keine gute Wahl, wenn es um die Herstellung differenzierender Software geht.

Für Unternehmen, die nicht-differenzierende Software benötigen, im Grunde genommen für alle Firmen, ist es sehr ratsam, einen Teil der Entwicklungsarbeit mit Hilfe von Open Source zu leisten, um so eine höhere ökonomische Effizienz zu erreichen. Die Teilnahme an Open-Source-Entwicklungsgemeinschaften sollte einen wichtigen Teil einer Gesamtstrategie eines jeden Unternehmens darstellen, die darin besteht, weniger Geld in nicht-differenzierende Software zu investieren und dafür mehr in die Entwicklung differenzierender Software.

10 Auch im Rahmen des Vertriebsmodells entwickelte Software beinhaltet Teilnahmekosten. Teilweise entsprechen diese den oben im Zusammenhang mit dem „Staubfänger"-Problem erwähnten Kosten. Keine Erwähnung finden die Kosten, die nach dem Erwerb entstehen. Hier finden sich unter anderem die Kosten für die Dokumentation durch den Händler (häufig eine Vollzeitaufgabe für die Programmierer) und die Kosten für die häufig langwierigen Verhandlungen mit der Serviceabteilung des Händlers, bis ein bestimmtes Problem behoben ist.

8 Wer trägt zu Open Source bei und wie finanziert sich dieser Beitrag?

Die Open-Source-Teilnehmer können folgendermaßen kategorisiert werden:

- Freiwillig Mitwirkende
- Anbieter von Linux-Distributionen
- Unternehmen mit nur einem Open-Source-Programm als ihrem Hauptprodukt
- Unternehmen, denen Open-Source-Software den Verkauf von Hardware und Lösungen ermöglicht
- Endanwender-orientierte Unternehmen und deren Auftragnehmer
- Dienstleistungsunternehmen
- Staatliche Institutionen und Verwaltungen
- Akademiker und Wissenschaftler

Freiwillig Mitwirkende

Ich (Bruce Perens) war selbst ein Beispiel für diese Form der Mitarbeit, als ich mich von 1993 bis 1998 intensiv mit der Entwicklung von Open Source beschäftigt habe, obwohl dies in keinerlei Verbindung zu meiner beruflichen Tätigkeit stand.

Die FLOSS-Studie des *International Institute of Infonomics* (2002) beschrieb die Entwicklung von Open Source zu jener Zeit in erster Linie als eine Art Freizeitbeschäftigung. Die Mitarbeit von Privatpersonen, die nicht direkt aufgrund einer primär finanziellen Motivation an der Entwicklung von Open-Source-Software mitwirken, ist beträchtlich genug, um über den Begriff des Hobbys hinauszugehen. Angesichts der offenkundigen Professionalität ihrer Arbeit[11] und der Tatsache, dass sie sich nicht aus pekuniären Gründen am Projekt beteiligen, ist freiwillig Mitwirkender vielleicht eine treffendere Beschreibung.

Schon Eric Raymond hat darauf aufmerksam gemacht, dass die Motivation der freiwilligen Mitarbeiter in erster Linie immaterieller Natur ist und dass ein wichtiger Beweggrund für ihre Beteiligung die Mitgliedschaft in einer Gemeinschaft gegenseitigen Respekts ist, in der die Entwickler von ihren Mitstreitern für die Qualität und den innovativen Charakter ihrer Arbeit Anerkennung erhalten. In der FLOSS-Studie wurden Open-Source-Entwickler hinsichtlich ihrer Motivation befragt, und so wurde herausgefunden, dass sich viele aus reiner Wissbegier und dem Bestreben, ihre Kenntnisse zu erweitern, engagieren. Ich denke, dass ihre Beweggründe mit denen eines

11 Eine Studie von *Coverity* in Zusammenarbeit mit der *Stanford University* (Kerner 2006) zeigt, dass die am häufigsten verwendeten Open-Source-Anwendungen 0,434 Bugs pro 1 000 Zeilen Code beinhalten, während nach Angaben der *Carnegie Mellon University* (Delio 2004) kommerzielle Nicht-Open–Source-Software eine Fehlerhäufigkeit von 20–30 Bugs pro 1 000 Zeilen Code aufweist.

Künstlers vergleichbar sind: Ebenso wie der Antrieb eines Malers sich darin manifestiert, dass die Menschen seine Gemälde bewundern, wünscht ein Programmierer sich Nutzer, die seine Software zu schätzen wissen.

Mit der zunehmenden Nutzung von Open Source innerhalb von Firmen, die sich dann im Laufe der Zeit für deren Weiterentwicklung engagieren, kommt es zu einem stetigen Wandel vom freiwillig zum professionell Mitwirkenden. Die einst freiwillig Mitwirkenden werden von Firmen angestellt, die ihre Mitarbeit an Open-Source-Projekten in der Arbeitszeit unterstützen. In dem Moment, da Arbeitgeber gegenüber freiwillig Mitwirkenden ein Interesse an Open Source bekunden, treten diese aus „ihrem stillen Kämmerlein" und werden zu Experten innerhalb des Unternehmens.

8.1 Anbieter von Linux-Distributionen

Red Hat und *Novell* sind als Distributoren Linux-basierter Systeme bekannt. Erstaunlicherweise wird die Mehrheit der Open-Source-Software jedoch nicht von Firmen, die Open-Source-Software als Hauptprodukt verkaufen, entwickelt. Sie verwerten im Wesentlichen nur die Arbeit anderer. Sie übernehmen Wartungsarbeiten, um Bugs für ihre zahlenden Kunden zu entfernen und sie beteiligen sich dann an der Entwicklung von Open-Source-Software, wenn dies für ihr Produkt oder die Erschließung eines neuen Marktes erforderlich ist. Bei den Linux-Distributionen handelt es sich meist um Unternehmen mittlerer Größe, deren Arbeitnehmer nur einem kleinen, wenngleich sehr aktiven Kreis der Open-Source-Gemeinschaft angehören. Gelegentlich übertreiben sie im Rahmen ihres Marketings ihre Bedeutung für die Entwicklung. Beispielsweise wirbt eine Linux-Distribution damit, 300 Programmierer zu beschäftigen, jedoch beteiligt sich nur ein viel geringerer Teil regelmäßig an der Entwicklung von Open-Source-Software. Der Großteil scheint sich vielmehr mit der „Vertriebstechnik" oder der Erarbeitung interner Lösungen für Kunden zu befassen.

Unter dem wirtschaftlichen Aspekt betrachtet, scheint Open Source für Linux–Distributionen eigentlich am wenigsten geeignet zu sein. Wenn man die Merkmale offenlegt, mit denen man sich von anderen Firmen differenziert, ermöglicht man der Konkurrenz, sich diese anzueignen und damit die Produktdifferenzierung zu schmälern. Genau dies ist denn auch die missliche Lage, in der sich die Linux-Distributionen befinden: Ihre Produkte sind hauptsächlich Open Source, die Kunden wünschen auch, dass es so bleibt, und so haben sie Schwierigkeiten zu erreichen, dass sich ihr Produkt von den anderen abhebt, insbesondere, da der Kunde weiß, dass das Produkt anderswo kostenlos angeboten wird.

Linux-Distributionen versuchten ursprünglich, dem Problem der Produktdifferenzierung mit den Open-Source-Verkaufsstrategien der ersten Generation beizukommen, die von Eric Raymond in „The Cathedral and the Bazaar" erläutert werden. Die Mehrzahl dieser Geschäftsmodelle sah eine Kombination von Open-Source-Software mit einem anderen Produkt als Einnahmequelle vor: Dienstleistungen für das Open-Source-Produkt oder eine proprietäre Softwareerweiterung der Open-Source–

Komponente. Die Open-Source-Software wäre hierbei die Geschäftsgrundlage des Unternehmens, die Einnahmen würden allerdings über andere Produkte generiert werden, wobei diese mit der Open-Source-Software unmittelbar im Zusammenhang stünden. Leider waren Dienstleistungen in der Zeit der *early adopters* für *Linux*[12] allein nicht ausreichend, um eine entsprechende Profitabilität der Distributionen zu erzielen.

Heute erfahren wir die zweite Generation der Open-Source-Verkaufsstrategien. Einige der Linux-Distributionen sind proprietärer Software nachempfunden. Hinter ihrem kostenpflichtigen Softwarepaket oder ihrer Per-Platz-Lizenzen verbirgt sich ein Produkt, das der Kunde anderswo kostenfrei bekommen könnte. Damit dieses Prinzip überhaupt funktioniert, wurden verschiedene Strategien miteinander kombiniert: Zunächst erfolgt die Entwicklung einer Marke, der mehr Vertrauen und Wert zugeschrieben wird als der Software an sich. Der Verkäufer investiert nun in die Zertifizierung des Produkts durch Anbieter proprietärer Anwendungen, von denen jeder nur für einige der Distributionen Supportleistungen zur Verfügung stellt.[13] Kunden, die Support für eines der proprietären Produkte auf Linux-Basis brauchen, wird so ein Anreiz gegeben, sich die Linux-Version dieses Anbieters zu kaufen. Einige dieser Supportleistungen sind allerdings nur eingeschränkt nutzbar, wie z. B. Berichte zu Sicherheitsproblemen und Bug-Patches, die nur für diejenigen verfügbar sind, die sich das kostenpflichtige Softwarepaket und die Einzelplatzlizenzen angeschafft haben. Wenn ein Kunde die freie Software auf mehr Systeme laden sollte, als er bezahlt hat und der Verkäufer das herausfindet, dann erfolgt die Aufhebung seines Wartungsvertrages und es werden ihm Sicherheitsinformationen vorenthalten, die wichtig für den weiteren reibungslosen Betrieb seiner Software wären. Die passendste Bezeichnung für dieses Geschäftsmodell ist daher wahrscheinlich proprietäre Open Source, bei der Support- und Serviceleistungen angeboten werden, das Geschäft jedoch im proprietären Bereich operiert. Dieses Geschäftsmodell widerspricht der eigentlichen Idee von Open Source und steht daher in antagonistischer Art und Weise dem Beitrag der freiwillig Mitwirkenden, die den Großteil der Open-Source-Software entwickelt haben, gegenüber. Diese hatten nicht die Absicht, ein zweites *Microsoft* zu schaffen und sperren sich folglich gegen die Beschlagnahmung von Serviceinformation ihrer Software. Solches steht in klarem Widerspruch zum Geiste, wenn auch nicht zum Wortlaut, von Lizenzmodellen wie der *GPL*. Generell unterstützen die freiwillig Mitwirkenden genau jene Unternehmen, von denen sie auch etwas halten. Als interne

12 Der Begriff *early adopter* (englisch für: frühzeitiger Anwender, Visionär) bezeichnet eine Firma, die frühzeitig die neusten technischen Errungenschaften oder die neusten Varianten von Produkten erwirbt. Die Zeit der *early adopters* ist Bestandteil der Diffusionstheorie und stellt die zweite Phase der Marktakzeptanz eines innovativen Produkts dar (Anm. d. Red.).

13 Die *Linux Standard Base* war ursprünglich dafür gedacht, proprietären Anwendungsanbietern die Möglichkeit zu geben, ihre Produkte für alle standardisierten Linux-Distributionen anzubieten. Sie ist inzwischen von der *ISO* als internationaler Standard akzeptiert worden und verschiedene prominente Linux-Distributionen (*RedHat*, *SUSE*, *Ubuntu* u. a.) bemühen sich um Zertifizierung.

Open-Source-Experten ihrer Firma empfehlen sie nur die Firmen weiter, die sie für gut befinden. Somit steht das Geschäftsmodell der proprietären Open Source vor nicht unbedeutenden Herausforderungen.

An dieser Stelle ist es wichtig, hervorzuheben, dass sich nicht alle Linux-Distributionen an proprietärer Open Source beteiligen und dass proprietäre Open Source bei der Entwicklung von Open-Source-Software nicht das übliche Geschäftsmodell ist. Die überwiegende Mehrheit der Mitwirkenden an Open Source gehört einer der anderen der hier untersuchten Kategorien an.

Das Geld für den Kauf von Linux-Distributions-Lizenzen und den damit verbundenen Support- und Serviceleistungen stammt aus den IT-Kostenstellen der Unternehmen.

8.2 Unternehmen mit nur einem Open-Source-Programm als ihrem Hauptprodukt

Diese Unternehmensform kann in verschiedene Modelle unterteilt werden:

- Modell der gemischten Lizenzierung: Open Source und proprietär
- Ein zentrales Open-Source-Programm in Verbindung mit proprietären Software-Accessoires
- Reines Open-Source-Modell mit den dazugehörigen Serviceleistungen

Im Folgenden wird jedes Modell näher betrachtet.

Modell der gemischten Lizenzierung: Open Source und proprietär

Beispiele hierfür sind Firmen wie *MySQL AB* und *Asterix PBX*, die Datenbanken erstellen, oder *Trolltech*, von denen das Qt-Toolkit für Programmierung graphischer Benutzerschnittstellen (GUI) produziert wird. Bei dieser Unternehmensform wird dieselbe Software unter zwei verschiedenen Lizenzen vertrieben: einer Open-Source–Lizenz und einer kommerziellen Lizenz.

Die gewählte Open-Source-Lizenz, häufig die *GPL*, enthält dabei eine „bittere Pille", welche die Herstellung von proprietären Folgeentwicklungen zu kommerziellen Zwecken verhindern soll. Da die *GPL* vorschreibt, dass Derivate generell nur als Quellcode im Rahmen der *GPL* oder einer zur *GPL* kompatiblen Lizenz vertrieben werden dürfen, würde man die Produktdifferenzierung jeder Software unterbinden, auf die dieses zutrifft.

Um dem aus dem Weg zu gehen und eine Produktdifferenzierung zu erhalten, muss der Hersteller proprietärer Derivate eine kommerzielle Lizenz für das gleiche Produkt erwerben. Dies verschafft dem Entwickler der Open-Source-Software ein direktes Einkommen für den Einzelverkauf von Software an die Hersteller solcher proprietärer Folgeentwicklungen.

Dieses Modell funktioniert allerdings nur bei Software, die in solche abgeleiteten Arbeiten integriert wird, etwa bei Softwarebibliotheken. Für Anwendungen ist dieses Modell im Allgemeinen unbrauchbar.

Die Zukunft dieses Modells ist jedoch fraglich, da ein Programmierer eine Softwarebibliothek als „Server" erstellen und alle darin enthaltenden Funktionen in ein anderes Programm exportieren kann, ohne dabei etwas zu schaffen, was nach dem Urheberrecht oder den Definitionen der Open-Source-Lizenzen als Derivat gelten würde. In der UNIX-Sprache werden solche Server als *daemons* bezeichnet, weshalb die Einbettung von Software in einen *daemon*, um die Herstellung eines Derivats zu vermeiden, als *daemonization* bezeichnet wird. Möglicherweise wird dieser Vorgehensweise durch zukünftige Open-Source-Lizenzen ein Riegel vorgeschoben.

Heutzutage kann man den MySQL-Datenbankserver in einer proprietären Anwendung nutzen, ohne über eine entsprechende Handelslizenz zu verfügen. Hierfür muss man lediglich die *MySQL Database Engine* als Server verwenden (Standardmodus) und sich einer speziellen Variante der *MySQL Client Library* bedienen, deren Lizenzvereinbarung für proprietäre Anwendungen geeignet ist. Eine solche Bibliothek wird von *MySQL AB* selbstverständlich nicht unterstützt.

Solcherart orientierte Unternehmen steigern üblicherweise ihre Einkünfte aus der kommerziellen Lizenzierung durch zusätzliche Einnahmen aus Schulungen und Softwareentwicklungen.

Die Hauptkunden von *MySQL* sind gewerbliche Nutzer. Trolltech zählt in der Hauptsache Anwendungsentwickler und Entwickler aus dem Embedded-Devices–Bereich zu seinen Kunden. Die Geldmittel für den Erwerb derartiger Produkte kommen zumeist aus den Budgets der IT- und Entwicklungsabteilungen.

Ein zentrales Open-Source-Programm in Verbindung mit proprietären Software-Accessoires

Eric Raymond nennt dieses Modell *widget frosting* und hat es ausführlich beschrieben. *Sendmail Inc.* ist ein Beispiel dafür. *Sendmail* hat eine Serie proprietärer Produkte rund um den Open-Source-E-Mail-Server *Sendmail* entwickelt. Über diese werden die anfallenden Pflege- und Wartungsarbeiten des zentralen Open-Source-Produkts finanziert. Einige Anbieter von Linux-Distributionen beabsichtigen, auf diesem Modell zu operieren, und so mancher IBM-Geschäftsbereich verwendet es bereits: z. B. der Vertrieb der proprietären DB2-Datenbank für *Linux*. Diese Unternehmen arbeiten im Grunde genau wie die Firmen, die proprietäre Software auf den Markt bringen. Die Geldmittel für den Erwerb der Produkte entstammen den Budgets der IT-Abteilungen.

Reines Open-Source-Modell mit den dazugehörigen Serviceleistungen

Dieses Modell wurde in Eric Raymonds Artikel als eines der wichtigsten beschrieben, hat sich aber bisher als weniger erfolgreich als erwartet erwiesen. Viele Open-Source–Entwickler bessern ihre Einkünfte auf, indem sie Serviceleistungen für die von ihnen entwickelten Programme anbieten. Auch eine Anzahl neuer Firmen hat dieses Modell bereits adaptiert, einige unter Verwendung erheblichen Risikokapitals. Viele dieser neuen Firmen scheinen ein Zertifizierungsmodell zu verwenden, bei dem sie Dienstleistungen für eine bestimmte, zertifizierte Version der Software anbieten und das vielleicht mit dem als proprietäre Open Source bezeichneten Geschäftsmodell einiger Linux-Distributoren vergleichbar ist.

Einige wenige der klein- und mittelständischen Unternehmen haben es geschafft, nachhaltige Einnahmen zu erzielen mit der Entwicklung von Open-Source-Software als ihrem zentralen Tätigkeitsbereich und der Serviceleistung für diese Software als ihrem Ertragszentrum. Viele sind jedoch auch spektakulär gescheitert, etwa *Linuxcare* (heute *Levanta*). Firmen, die eigentlich potenzielle Servicekunden wären, scheinen es vorzuziehen, ihren Open-Source-Support intern zu regeln bzw. über einen Anbieter abzuwickeln, zu dem sie bereits Geschäftsbeziehungen unterhalten oder der in der Lage ist, Serviceleistungen für mehr als ein Programm zur Verfügung zu stellen. Darüber hinaus kann hier aber auch ein Early-Adopter-Problem vorliegen. Solche Erstanwender lehnen es im Allgemeinen ab, von einem Serviceunternehmen „bemuttert" zu werden. Es wird sich erst noch zeigen, ob dieses Modell effektive Ergebnisse erzielen kann.

Die Geldmittel für die Serviceleistungen, die von einem solchen Unternehmen zur Verfügung gestellt werden, stammen auch hier aus den Budgets der IT-Abteilungen.

8.3 Unternehmen, denen Open-Source-Software den Verkauf von Hardware und Lösungen ermöglicht

IBM und *HP* sind beispielsweise solche Unternehmen. Hardware lässt sich großartig in Verbindung mit Open-Source-Software verkaufen. Es kostet bekanntlich fast nichts, Software zu kopieren, jedoch kann man ein Brot nicht vervielfachen, ohne zumindest ein Pfund Mehl zu verbrauchen. Solange wir nicht im Besitz des „Star–Trek-Replikators"[14] sind, wird Hardware ein schwer zu vervielfältigendes Produkt bleiben. Wenn man dem Kunden Einblick in die Interna eines Hardwareprodukts gewährt, bedeutet dies nicht zwangsläufig, dass man dessen Produktdifferenzierung dadurch aufhebt, wie das vielleicht bei Software der Fall wäre. Die Hersteller von Hardware, die bei der Entwicklung von Open Source mitwirken, tun dies, um den Absatz

14 Der *Replikator* ist ein fiktionales Gerät, der viele verschiedene physische Objekte auf Abruf produzieren kann und dabei vermutlich auf in einem Computer abgelegte Designs zurückgreift. In der Serie *Star Trek: The Next Generation* trat eine Person an das Gerät heran und gab ihm eine verbale Anweisung: „tea, earl grey, hot." Eine Tasse und der Tee wurden im Bruchteil einer Sekunde produziert und bereitgestellt.

ihrer Hardwareprodukte zu fördern. Denn Hardware ist ohne die entsprechende Software nutzlos und dies gilt insbesondere für Computer, die ohne ein Betriebssystem, das Computerhardware und Softwareanwendungen miteinander verbindet, nicht zu gebrauchen wären. Die Entwickler von Open-Source-Software scheinen vor allem auf dem Gebiet der Systemprogrammierung besser zu sein als auf jedem anderen Gebiet der Programmierung. Der Linux-OS-Kernel ist inzwischen mindestens genauso gut, wenn nicht besser, als viele proprietäre Betriebssysteme für vergleichbare Hardware. Die Hersteller von Hardware haben früher Milliarden für proprietäre Betriebssysteme ausgegeben, die für sie immer eher technologische Notwendigkeit als Ertragszentrum waren. Die Gewinnspannen wurden durch die Hardware bestimmt. Viele der Hersteller haben *Linux* bereitwillig übernommen, weil es ihnen die Verteilung von Kosten und Risiken des Betriebssystems auf mehrere Firmen ermöglicht, die Rentabilität zudem größer ist als bei vergleichbaren proprietären Betriebssystemen und es obendrein bei den Kunden hoch im Kurs steht.

Hardwareunternehmen eignen sich sehr gut als Open-Source-Produzenten, weil Open Source und Produktdifferenzierung in diesem Falle nicht so stark miteinander in Konflikt stehen wie bei Softwareunternehmen. Softwareprodukte offenzulegen, ermöglicht diesen Unternehmen den Vertrieb, mindert nicht im Geringsten die Differenzierung ihrer Hardware-basierten Produkte und kann somit ihr Ertragszentrum nicht negativ beeinflussen.

Die Erlöse aus dem Hardwarevertrieb werden bei dieser Art von Unternehmen üblicherweise durch Serviceleistungen und mitunter auch Schulungen erhöht. Die finanziellen Mittel für die Gesamtheit der angebotenen Hardware und Serviceleistungen stammen aus den jeweiligen Hardwarebudgets der IT-Abteilungen.

8.4 Endanwender-orientierte Unternehmen und deren Auftragnehmer

eBay ist ein Beispiel für diese Art der Beteiligung. Open-Source-Software wird von vielen Firmen für ihre eigenen Arbeitsabläufe genutzt, wobei Webapplikationen, neben vielen anderen, einen besonderen Stellenwert haben. Diese Unternehmen bilden einen signifikanten Teil der Mitwirkenden an Open Source. Ihre Beiträge stammen im Allgemeinen von den Mitarbeitern des Softwaresupports und der Softwareentwicklung der Firmen oder aber ihren Dienstleistern, welche die Open-Source-Software den Bedürfnissen des Unternehmens entsprechend anpassen. Das Interessante an diesen Unternehmen ist, dass sie im Grunde genommen die Hauptklientel aller anderen Unternehmen, die zur Entwicklung von Open-Source-Software beitragen, ausmachen. Aus ihren Budgets wird die Arbeit der anderen hier aufgeführten Mitwirkenden finanziert. Zusätzlich scheinen sie einen Teil ihrer Gelder ohne die üblichen Umwege direkt in die Entwicklung von Open-Source-Software zu investieren, sei es nun über die Arbeit der eigenen Mitarbeiter oder aber über ihre direkten Auftragnehmer.

Doch warum sollte ein solches Unternehmen außerhalb seiner Kernkompetenzen überhaupt einen Beitrag leisten? Da man den Ausdruck Kernkompetenzen eher auf

Individuen als auf Unternehmen bezieht, sollte man nicht darüber nachdenken, ob ein Open-Source-Projekt im Mittelpunkt der eigenen Geschäfte steht, sondern eher darüber, ob die optimale Wirtschaftlichkeit des Unternehmens dadurch erreicht wird, dass die Beteiligung am Open-Source-Projekt zur Hauptaufgabe einiger Mitarbeiter erhoben wird. Die entscheidenden Vorteile sind in diesem Fall die Kosten- und Risikosenkung und gleichzeitig die Verbesserung der Kontrolle, den man über die eigene Software hat. Sie sollten sich deshalb einmal vor Augen führen, in welchem Umfang Software Ihre Geschäftsabläufe kontrolliert. Haben Sie Kontrolle über Ihre Software?

8.5 Dienstleistungsunternehmen

Eine Reihe von Dienstleistungsunternehmen entwickelt Lösungen, bei denen verschiedene Open-Source-Programme mit anderer, direkt auf den jeweiligen Kunden zugeschnittener, Software als „Bindeglied" kombiniert werden. Andere bieten für eine bestimmte Anzahl an Open-Source-Programmen entsprechende Serviceleistungen an. Diese Unternehmensform beteiligt sich an der Entwicklung und Pflege verschiedener Open-Source-Programme, dabei jedoch selten in größerem Umfang für ein spezielles Programm. Im Allgemeinen kann man sagen, dass das Interesse des Auftraggebers das solcher Auftragnehmer häufig übersteigt, weshalb diese Dienstleister oft in die obige Kategorie Endbenutzerunternehmen und deren Auftragnehmer fallen. Einige Firmen bieten Webdienste auf Basis des Application-Service-Provider-Modells (ASP) an und bauen dabei hauptsächlich auf Open-Source-Software. Open-Source-Software hauptsächlich deswegen, weil es in vielen Open-Source-Lizenzen (insbesondere der derzeitigen Fassung der *GPL*) ein Schlupfloch gibt, das die Differenzierung solcher Unternehmen schützt. Die aus der Lizenz resultierende Pflicht, den Quellcode einer Software offenzulegen, wird erst durch die Weitergabe der Software begründet – die ASPs betreiben die Software für den Kunden jedoch lediglich und müssen sie nicht direkt zur Verfügung stellen. Da diese Lücke häufig als Makel der *GPL* angesehen wird, ist es möglich, dass sie in der nächsten Version dieser Lizenz nicht mehr existiert.

8.6 Staatliche Institutionen und Verwaltungen

Die Art und Weise, wie Open Source von staatlichen Institutionen und Verwaltungen genutzt wird, unterscheidet sich kaum von der eines Wirtschaftsunternehmens. Anders als bei einem Unternehmen wird von der Verwaltung jedoch erwartet, dass sie sich für das Wohl ihrer Bürger einsetzt, weshalb man normalerweise nicht davon ausgeht, dass die Verwaltung so etwas wie ein „Firmenkonto", auf dem Gewinne verbucht werden, besitzt. Sie stellt also eher Dienstleistungen zur Verfügung, welche die Wirtschaft ankurbeln und gesellschaftliche Aktivitäten unterstützen.

Ein Auftrag aus der Verwaltung sollte einem einzelnen Anbieter, abgesehen von den direkten Einnahmen für Produkte und Dienstleistungen, die aus dem Auftrag selbst re-

sultieren, keinesfalls einen wirtschaftlichen Vorteil verschaffen. Die Verwaltung sollte sich zur Vermeidung von *lock-in* und den damit verbundenen Wechselkosten (*switching costs*) nie an einen einzelnen Händler binden. Es gilt deshalb als schlechte Politik, wenn die Anbieter und Bürger auf ein einzelnes Produkt festlegt werden, nur um mit der Verwaltung in Kontakt zu treten, denn das würde dem Händler einen unangemessenen Vorteil verschaffen.

Jeder Händler kann im Rahmen der entsprechenden Lizenz von Open Source Gebrauch machen. So kann garantiert werden, dass die Schnittstelle zu den Einrichtungen der Verwaltung offengelegt wird und von allen gleichermaßen genutzt werden kann. Im Ergebnis kann mittels der Verwendung von Open Source ein E-Government–Modell zur Kommunikation zwischen Verwaltung und Bürgern, zwischen Verwaltung und Unternehmen und zwischen Verwaltungsapparaten untereinander ermöglicht werden.

Üblicherweise werden von der Verwaltung dem Gemeinwohl zuträgliche Aufgaben erfüllt. Die Open-Source-Entwicklungen können dann als Teil derselben durchgeführt werden. Dies erfolgt im Allgemeinen durch die Vergabe von Forschungsgeldern.

8.7 Akademiker und Wissenschaftler

Akademische Forschungsprojekte haben von jeher einen erheblichen Beitrag zur Entwicklung von Open Source geleistet, dies trifft insbesondere auf die Arbeit im Hauptstudium – eher noch als im Grundstudium – zu. Grundstudiumskurse geben den Studenten nicht genügend Zeit, sich in ein Open-Source-Projekt einzuarbeiten und selbst einen Beitrag zu leisten. Im Gegensatz dazu laufen Forschungsprojekte im Hauptstudium meist über mehrere Jahre. Ein großer Teil der Open-Source-Software wurde vom BSD-Projekt (*Berkeley System Distribution*) entwickelt, das vom *U.S. Department of Defense* finanziert wurde. Auch andere Forschungsprojekte von Studenten entwickeln Tag für Tag immer neue Software.

Es sind energische Open-Source-Communitys, die sich an solchen wissenschaftlichen Forschungen beteiligen. In der Wissenschaft gilt die Maxime *publish or perish*[15] und Open Source wird dieser Maxime gerecht. Um als stichhaltig zu gelten, muss wissenschaftliche Forschungsarbeit replizierbar sein. Wenn ein Experiment, wenn es von einem anderen Wissenschaftler durchgeführt wird, nicht die gleichen Ergebnisse erzielt, dann kann dies ein Hinweis darauf sein, dass die Forschungsarbeit fehlerbehaftet ist. Dieser Tage bildet Software einen großen Bestandteil vieler Experimente, die menschliche Sprache zur Beschreibung eines Experiments vermag jedoch nicht jedes Detail seines Ablaufs zu vermitteln. Wenn Wissenschaftler ihren Quellcode anderen

15 *Publish or perish* (dt. „veröffentliche oder gehe unter") ist eine im Wissenschaftsbetrieb, insbesondere an Universitäten, gängige Redewendung, mit der zum Ausdruck gebracht werden soll, dass Forscher einem starken informellen Druck ausgesetzt sind, ihre Ergebnisse möglichst zahlreich und in möglichst angesehenen Verlagen oder Fachzeitschriften zu veröffentlichen, um ihr wissenschaftliches Renommee zu steigern.

zugänglich machen, dann können Außenstehende ihre Software auf mögliche Fehler hin überprüfen und die Experimente leichter nachvollziehen.

Die Anzahl der Wissenschaftler, die auf einem bestimmten Forschungsgebiet tätig sind, ist relativ klein. Das Modell der Softwareentwicklung zu Vertriebszwecken wäre ungeeignet für die Entwicklung derart spezieller Software. Open-Source-Software ist insofern der beste Weg für Wissenschaftler, die Kosten für die Entwicklung von Software zur Unterstützung ihrer Forschungsarbeiten zu verteilen.

Befürworter proprietärer Software haben versucht, den großen Anteil wissenschaftlicher Beiträge an Open Source zu verringern, indem sie Partnerschaften zwischen Forschungsprojekten an Universitäten und proprietären Softwareunternehmen schlossen. Dieser Trend erscheint insbesondere dann beunruhigend, wenn staatlich finanzierte Forschungsarbeiten in Patenten resultieren, die bei den Partnerunternehmen, den Herstellern proprietärer Software, liegen. Beunruhigend deshalb, weil es sehr wahrscheinlich ist, dass diese Patente eines Tages dem Steuerzahler gegenüber, der ihre Entstehung ursprünglich finanziert hat, rechtlich durchgesetzt werden. Generell sollte staatlich finanzierte Forschungsarbeit bis ins kleinste Detail der Öffentlichkeit nutzbar gemacht werden, denn sie hat die Arbeit ja finanziert. *DARPA* (*U.S. Department of Defense Research Grant Organization* – die US-Vergabestelle von Forschungsstipendien des Verteidigungsministeriums) und die *University of California* haben das erkannt, als sie die BSD-Lizenz auf die ersten UNIX-Erweiterungen, die an der Universität entwickelt wurden, anwendeten. Diese Lizenzierung erlaubte sowohl eine Nutzung als Open Source als auch als proprietäres Produkt.

Ein Teil der Forschungsarbeit wird von Studenten ohne ein entsprechendes Entgelt betrieben. Wenn ihre Tätigkeit allerdings vergütet wird, dann stammt das Geld für die Entwicklung der Open-Source-Software meist aus dem Budget für Forschungsstipendien. Es sind in erster Linie staatliche Organe, die solche Stipendien vergeben. Ein Teil wird aber auch durch gemeinnützige Organisationen oder von Industriepartnern finanziert.

8.8 Open-Source-Mitwirkende im Überblick

Den größten Teil zu der Entwicklung von Open Source tragen immer noch die freiwillig Mitwirkenden bei. Zwischen dem unternehmerischen Bestreben nach Differenzierung und Open Source besteht ein Konflikt, dessen ökonomische Nachteile am stärksten bei jenen Unternehmen zum Tragen kommen, die normalerweise in die Entwicklung proprietärer Software investieren würden. Unternehmen, die dieser Konflikt nicht betrifft, können effektiver zur Entwicklung von Open Source beitragen. Hardwareherstellern kommt aufgrund dessen eine bedeutende Funktion zu. Auch Endanwender-orientierte Unternehmen und deren Auftragnehmer spielen eine immer bedeutendere Rolle.

9 Wie das Produktmarketing bei Open Source funktioniert

Kritiker von Open Source führen an, dass die Entwicklung von Open Source nicht fokussiert und zielgerichtet genug ist. Für gewöhnlich obliegt die Entwicklung eines Produktes einem Unternehmen, das seine Geschäftstätigkeit, angetrieben durch einen definierten Produktmarketingprozess, auf dieses Produkt ausgerichtet hat. Einige fortgeschrittene Open-Source-Projekte betreiben Produktmarketing, das mit dem eines Unternehmens vergleichbar wäre. Aber während ein Unternehmen eine Strategie entwickelt, welche die gesamte Softwareentwicklung bestimmt, gibt es für Open Source keine überregionale Planungsinstanz, die eine umfassende Strategie festlegt, an der sich die Open-Source-Entwickler orientieren. Die Aussage ist jedoch in sich falsch, insofern, als dass Open Source nicht mit einem Unternehmen vergleichbar ist. Es beschreibt eine gesamte Industrie. Eine zentrale Planungsinstanz für eine solche Industrie würde jedoch alles andere als einen offenen Markt darstellen. Das Produktmarketing der globalen Open-Source-Community orientiert sich daher eher an den Funktionsweisen der kapitalistischen Marktwirtschaft als an der Art, wie ein einzelnes Unternehmen seine Produkte planen würde.

Die Methoden, nach denen Open Source Produkt-Marketing betreibt, können als vielfach parallele Zufallsbewegungen, die in einem darwinistischen Prozess gefiltert werden, beschrieben werden. Zunächst entwickeln viele Menschen auf der ganzen Welt die Dinge, die sie selbst benötigen und schlagen dabei Wege ein, die sie sich selbst aussuchen – ohne jegliche zentrale Koordination. Auf diese Art entstehen die unterschiedlichsten potenziellen Produkte; ein Großteil von ihnen ist dabei nur für eine Handvoll anderer interessant. Es entstehen aber auch Produkte, für die sich mehr als fünfzig andere Personen interessieren und es entstehen einige wenige, die für Millionen von Menschen interessant sind.

Fünfzig Personen, die sich für ein und dieselbe Sache interessieren, aber über die ganze Welt verteilt sind, können über das Internet ein funktionierendes Softwareentwicklungsteam bilden. Die freie Zeit von fünfzig Personen, die beruflich anderweitig beschäftigt sind, erweist sich als ausreichend für die Entwicklung großer und komplexer Softwareprodukte. Selbstverständlich wächst das Entwicklerteam in der Anzahl der Beteiligten, während das Produkt reift und mehr Menschen sich dafür interessieren. So kann man unter Ausnutzung der exzellenten Zusammenarbeit, die im Rahmen von Open-Source-Projekten möglich ist, Projekte ins Leben rufen und erfolgreich durchführen, die sonst die Möglichkeiten der beteiligten Parteien übersteigen würden.

Aber wie ist es möglich, dass fünfzig Personen, die einander noch nie getroffen haben, zusammenarbeiten und ein funktionierendes Softwareprodukt entwickeln können? Einen Erklärungsansatz für die gute Zusammenarbeit liefert die Tatsache, dass Software von Natur aus sehr modular aufgebaut ist und folglich viele Personen an unterschiedlichen Segmenten der Software fast autonom arbeiten können. Sie müssen sich letztendlich nur über die Zusammenfügung der einzelnen Teile einigen. Ein

gutes Beispiel dafür ist die Debian-GNU/Linux-Distribution. Sie beinhaltet ungefähr 16 000 Programmpakete, die von mehr als 1 000 freiwilligen Entwicklern aus vielen Ländern der Welt bereitgestellt werden. Zusammengefügt stellen die Programmpakete ein zuverlässiges und gut integriertes System dar. Ein System, das Experimente in einer Weltraumfähre in der Erdumlaufbahn überwacht hat und eine Anwendergemeinschaft besitzt, die es in den Top-Fünf aller Distributionen platziert.

Aber wie ist es möglich, Open-Source-Software zu entwickeln, wenn jedem Entwickler maximaler Freiraum zugestanden wird und es keine wirkliche Autorität gibt? Das erscheint Geschäftsleuten merkwürdig, bis sie darauf kommen, dass genau so der Kapitalismus in demokratischen Ländern funktioniert. In der Wirtschaft eines kapitalistischen Staates wollen viele Unternehmen ihre Produkte ohne die Einmischung der Regierung entwickeln. Sie alle stehen zueinander im Wettbewerb und so entstehen Produkte, die sich erfolgreich am Markt etablieren können und andere, die versagen. Unternehmen arbeiten mitunter auch lose zusammen, wenn es darum geht, mit neuen Produkten neue Märkte zu erschließen. Die Rolle des Staates, der einzigen Institution, welche die Macht hätte, die Wirtschaft zu steuern, beschränkt sich im Allgemeinen darauf, der Wirtschaft Kapital zuzuführen bzw. zu entziehen, Steueranreize und Subventionsprogramme zu initiieren und die Einhaltung der Gesetze zu überwachen, die einem gerechten Marktgeschehen dienen sollen.

Die meisten Menschen akzeptieren die Tatsache, dass es so etwas wie eine „sichere Sache" im Spiel oder an den Aktienmärkten nicht gibt, und doch erwarten sie von Marketingabteilungen zuverlässige Prognosen, die der Entwicklung neuer Produkte dienlich sein sollen. Marketingabteilungen haben keine Kristallkugel. Wenn genügend viele selbstständige Entwickler vorhanden sind, können diese mit einer auf Zufallsbewegungen beruhenden Strategie erfolgreicher sein als konventionelles Produktmarketing. Im Rahmen der oben beschriebenen, vielfach parallel ablaufenden Zufallsbewegungen wird eine große Anzahl unterschiedlicher Wege beschritten, somit steigen die Chancen, einen erfolgreichen Weg einzuschlagen. Die darwinistische Filterung begründet dabei die Erkenntnis, ob ein bestimmter Weg erfolgreich ist.

Die meisten Geschäftsleute stehen fest hinter der freien Marktwirtschaft und einer Begrenzung des Einflusses, den der Staat auf das Was und das Wie der Produktion ausübt. Sie haben verstanden, dass ein freier Markt eher gute Produkte und eine gesunde Wirtschaft hervorbringen kann als eine zentrale Planwirtschaft. Es sollte sie daher nicht überraschen, dass das auf dem offenen Markt basierte Open-Source-Prinzip in der Schaffung wünschenswerter Produkte bessere Erfolge erzielen kann als zentral geleitete Marketing-Abteilungen und ihr Management.

10 Ist Open Source autark finanzierbar?

Vielen Menschen fällt es schwer zu glauben, dass Open-Source-Software sich selbst tragen kann, obwohl sie nicht in üblicher Weise als Vertriebsentwicklung entsteht. Wie

finanziert sich solcherart Software? Sie wird direkt oder indirekt aus den Budgets der Unternehmen finanziert, die sie benötigen. Diese Unternehmen wenden einen Großteil ihres Budgets für nicht-differenzierende Software auf. Sie sind deshalb gewillt, an einer Open-Source-Entwicklung dieser Software mitzuwirken, da durch die Verteilung der Kosten und Risiken auf viele Mitwirkende die diesbezüglichen Kostenzentren reduziert werden können und die Ausgaben für Software effektiver genutzt werden als bei vertriebsorientiert entwickelten Produkten. Dies sind in der Hauptsache die gleichen finanziellen Mittel, aus denen die Entwicklung proprietärer Software schöpft. Es ist wichtig, sich vor Augen zu führen, dass der Hauptfinanzier der Entwicklung eines Softwareprodukts nicht der Hersteller ist, sondern der Kunde.

10.1 Welchen Einfluss hat Open Source auf die Wirtschaft?

Wir haben den großen wirtschaftlichen Einfluss *Microsofts* wie folgt erklärt: Es ist *Microsoft* gelungen, viele Unternehmen (seine Kunden) leistungsfähiger in der Abwicklung ihrer Geschäfte zu machen. Hinzu kommt, dass diese Unternehmen ohne die Software von *Microsoft* bestimmte Geschäftsbereiche nicht mehr abdecken könnten. Gleiches trifft auch auf Open Source zu.

Auf Open Source basieren heute die meisten Webserver, der hauptsächliche E-Mail–Verkehr sowie Anwendungen in Unternehmen, Organisationen und im privaten Bereich. Somit kann Open Source auf wirtschaftliche Ergebnisse von etlichen Milliarden Dollar verweisen.

Jede technologische Verbesserung, die bewirkt, dass Geschäfte noch effizienter ablaufen können, bedeutet auch für die Wirtschaft eine Verbesserung. Open Source ermöglicht es Unternehmen, weniger für Software auszugeben und höhere Qualität sowie bessere Kontrolle über ihre Software zu erreichen. Das Geld, das an der Software eingespart werden kann, verschwindet nicht, es kann für wichtigere Dinge ausgegeben werden.

10.2 Was bedeutet dies für die Hersteller von Software?

Diese Diskussion wurde eher aus der Perspektive der Geschäftsanwender, die letztlich für die Softwareentwicklung aufkommen, als aus der Perspektive der Produzenten geführt. Aber was bedeutet das für die Hersteller von Software? Verkaufen diese fertige Software? Einige Unternehmen verkaufen eher den Prozess der Entwicklung als die Software an sich. Dies gilt im Allgemeinen für Unternehmen, die Beratungsleistungen und Lösungen anbieten. Ihre Kunden werden bereit sein, für die Entwicklung von Open-Source-Software zu zahlen.

Für ein Unternehmen, das stärker daran interessiert ist, Software zum Verkauf herzustellen, als dem Kunden Programmier- und Schulungsleistungen anzubieten, wird Open Source nur begrenzt Umsatz generieren.

Es kann durchaus sein, dass Open-Source-Software irgendwann ein Absinken der Nachfrage nach proprietärer Software verursacht. Dies bedeutet jedoch nicht, dass der Bedarf an Programmierern sinkt, denn Software wird nach wie vor ein gefragtes Gut sein. Freigewordene Programmierer proprietärer Software würden in Unternehmen abwandern, die Open-Source-Software wirtschaftlich erfolgreich produzieren können.

Es ist möglich, dass Programmierer, die in einem weniger unternehmerisch-orientierten Umfeld zu arbeiten anfangen, weniger verdienen. Allerdings sind die Gewinnüberschüsse proprietärer Softwareunternehmen bislang eher dem Management als der Belegschaft zugute gekommen. Es war bisher auch selten der Fall, dass Programmierer überraschende Gewinne aus den Belegschaftsaktien eines Unternehmens erzielt haben, das hauptsächlich Software für den Vertrieb entwickelt.

10.3 Welche Auswirkungen hat das Trittbrettfahrerproblem auf Open Source?

Das Trittbrettfahrerproblem ist in der Wirtschaft bekannt. Wie geht man mit Menschen um, die Vorteile aus einem Produkt oder einer Dienstleistung ziehen, ohne dem Anbieter eine Gegenleistung zu erbringen?

Alle Open-Source-Nutzer sind zunächst Trittbrettfahrer. Sie laden Software herunter, testen sie und setzen sie eventuell auch ein. Anfangs ziehen die meisten es nicht in Betracht, einen Beitrag zur Weiterentwicklung der Software zu leisten. Das ist erst dann der Fall, wenn sie die Software bereits gebrauchen und zusätzliche Funktionalitäten benötigen.

Wenn sie ein zusätzliches Feature benötigen, kann es sein, dass sie, anstatt einen der ursprünglichen Entwickler dafür zu bezahlen, die Software selbst um die benötigten Funktionalitäten erweitern. Kann man sie zu diesem Zeitpunkt noch als Trittbrettfahrer bezeichnen? Nein. Unternehmen, die einem Open-Source-Projekt als Entwickler beitreten, leisten einen Beitrag zu diesem Produkt und alle beteiligten Unternehmen profitieren in wirtschaftlicher Hinsicht, indem sie die Software in ihren Kostenzentren einsetzen. Die Interessen der einzelnen Entwickler von Open-Source-Software sind im Allgemeinen ähnlich, weil sie sich selbst für die Weiterentwicklung eines bestimmten Softwareprodukts entschieden haben, das für sie nutzbringend ist. Die Beiträge eines jeden Entwicklers nutzen dann auch den anderen Entwicklern.

Es gibt Entwickler, die ihre Motivation nicht aus dem Bestreben beziehen, eine Software für einen Unternehmensbereich bereitzustellen. Es sind Personen, die sich hauptsächlich aus schöpferisch-gestaltenden Beweggründen oder der wissenschaftlichen Forschung wegen beteiligen.

Freiwillig Mitwirkende empfinden die Tatsache, dass die von ihnen entwickelte Software anderen Nutzen bringt, als befriedigend, wie auch Künstler die Wertschätzung für ihre Bilder als befriedigend empfinden. Diese Freiwilligen profitieren von den Anwendern ihrer Produkte in immaterieller Hinsicht, sie stellen für sie die Motivationsquelle dar. Daher sollten auch solche Anwender nicht als Trittbrettfahrer angesehen werden.

Unternehmen, die einem bestimmten Open-Source-Produkt besondere Bedeutung beimessen, neigen dazu, Entwickler zu beschäftigen, die bereits ein gewisses Ansehen als Entwickler dieses Produkts gewonnen haben. Auf diese Weise tendieren Personen, die ohne finanzielle Interessen in ein Open-Source-Projekt eingestiegen sind, dazu, Beschäftigung in einem Unternehmen zu finden, das an dem Open-Source-Produkt ein Interesse hat. Als Resultat profitieren an der Entwicklung von Open Source Mitwirkende häufig in finanzieller Hinsicht von ihrer Teilnahme. Diese Tatsache stellt einen weiteren Grund dafür dar, dass diese Beteiligten den Anwender nicht als Trittbrettfahrer sehen sollten.

In der Wissenschaft profitiert man von einem konstanten Wissensaustausch, ähnlich dem einer Open-Source-Community, begründet dadurch, dass Wissenschaft dann am schnellsten voranschreiten kann, wenn Entdeckungen anderen Wissenschaftlern zugänglich gemacht werden und diese dann ihre Ideen beisteuern. Wissenschaftler erzielen den Lohn ihres Schaffens aus der Veröffentlichung ihrer Arbeit, weil diese ihr Ansehen bei anderen Wissenschaftlern erhöht und üblicherweise den Erfolg ihres beruflichen Werdegangs begründet. Wissenschaftler nutzen routinemäßig Open-Source-Software, um die softwarebezogenen Komponenten ihrer Arbeit zu veröffentlichen. Wissenschaftler sind ferner in ihrer Arbeit dadurch motiviert, der Gesellschaft einen Nutzen zu bringen und hegen ein Interesse daran, dass andere Menschen vom Gebrauch ihrer, durch wissenschaftliche Forschung entwickelten, Software profitieren. Für die akademischen und wissenschaftlichen Mitwirkenden an Open Source sind die Anwender ihrer Software somit von Nutzen und sollten nicht als Trittbrettfahrer gelten.

Letztlich sind die Nutzer von Open-Source-Produkten Menschen, die wir als aktive Teilnehmer in Open-Source-Projekten gewinnen können. Vergleichbar ist dies mit dem Bemühen der Händler, die breite Masse dazu zu bewegen, ihre Produkte zu kaufen. Bei einigen Anwendern geht unser Konzept stets auf.

An dieser Stelle stellt sich die Frage, ob dem Trittbrettfahrer-Problem in Bezug auf Software die gleiche Bedeutung zukommt wie für andere Arten von Produkten und ob es für Open Source überhaupt eine Rolle spielt. Jemand, der schwarz mit dem Bus fährt und die begrenzte Ressource Sitzplatz darin beansprucht, nimmt unter Umständen einem zahlenden Fahrgast die Möglichkeit, im Bus mitzufahren. Jemand, der sich illegal eine Kopie von *Microsoft Windows* hergestellt hat, beeinflusst unter Umständen die Nachfrage des Marktes, an dem Windows-Kopien gegen Bezahlung gehandelt werden. Er nutzt aber kein begrenztes Gut und schließt damit andere Windows-Nutzern aus. Ein Trittbrettfahrer, der Open Source benutzt, hat keinen negativen Einfluss auf irgendeinen Markt und verbraucht auch keine beschränkt verfügbare Ressource.

All dies lässt mich schlussfolgern, dass das Trittbrettfahrer-Problem der Open–Source-Entwicklung keinen wesentlichen Schaden zufügt.

10.4 Wenn das Open-Source-Prinzip funktioniert, warum baut sich dann nicht jeder sein eigenes Auto?

Das Open-Source-Modell funktioniert gut für Produkte, deren Gestaltung hauptursächlich für ihren Wert ist. Es ist bis heute erfolgreich eingesetzt worden, um Software zu entwickeln, eine Enzyklopädie zu erstellen sowie integrierte Schaltungen zum Einsatz in programmierbaren Halbleitern[16] zu entwerfen.

Die meisten Dinge des Lebens sind jedoch keine Software. Es kostet kaum etwas, eine Software zu kopieren. Ein ordentliches Brot herzustellen, kostet zumindest ein Pfund Mehl. Jemand muss den Weizen anbauen, diesen zu Mehl verarbeiten und all diese Anstrengungen müssen bezahlt werden.

Autos sind in ihrer Herstellung selbstverständlich viel komplexer als Brot, es wird eine große Anzahl physischer Prozesse durchlaufen und sehr teure Hilfsmittel zur Fahrzeugherstellung kommen zum Einsatz. Betrachtet man, dass zur Herstellung eines Elektromotors Erz gewonnen und zu Metall verarbeitet, Drähte gezogen, Bleche gepresst, Lager gegossen und bearbeitet und alle Teile präzise zusammengesetzt werden müssen, so ist es kein Wunder, dass für die Produktion von Autos eine ganze Industrie benötigt wird. In Kontrast dazu kann eine Einzelperson allein ein wichtiges Softwareprodukt erstellen.

Wenn es eines Tages möglich ist, dass wir komplexe dingliche Produkte erstellen können, indem wir ihr bloßes Erscheinungsbild festlegen und die Herstellung aus leicht erhältlichen Materialien und Elektrizität dann einer Maschine überlassen,[17] wird sich die Wirtschaft radikal verändern. Heute beschränkt sich unser Wirken noch auf das Produzieren einzelner Teile mit Hilfe von computergesteuerten Fräsmaschinen, einem langsamen und schmutzigen Produktionsprozess, der zudem noch immer manuelle Intervention benötigt. Eine gesunde Open-Source-Community befasst sich mittlerweile mit solchen Maschinen, und es sind auf diesem Gebiet die ersten gemeinsam entwickelten Werkstückentwürfe zu beobachten. Trotz dieser Entwicklungen muss die Forschung rund um die computergestützte Produktion noch erhebliche Fortschritte machen, damit wir eines Tages in unseren „Open-Source-Autos“ unterwegs sein können.

11 Zusammenfassung

Open Source finanziert sich selbst und funktioniert wirtschaftlich gesehen nach dem kapitalistischen Prinzip. Man muss keine Magie bemühen, um es wirtschaftswissenschaftlich zu erklären. Open Source ist ein extrem profitabler Bestandteil der freien

16 Die Entwürfe integrierter Schaltungen werden dabei in programmierbaren Halbleitern (*field programmable gate arrays* – FPGA) eingesetzt, speziellen elektronischen Bausteinen, die auf Basis eines solchen Entwurfs programmiert werden und sich daraufhin wie die entsprechende Schaltung verhalten. Hierbei handelt es sich um ein Beispiel, bei dem Hardware ähnlich wie Software verwendet wird.

17 Vergleiche die Ausführungen zum „Replikator“ unter obiger Fußnote 14.

Marktwirtschaft, da die vielen Menschen und Unternehmen, die auf das Modell aufbauen, dadurch in die Lage versetzt werden, einen ökonomisch relevanten Beitrag zu leisten. Es ist die *effizienteste* Methode der Softwareentwicklung, wenn es um die Entwicklung von nicht-differenzierenden Produkten geht, die das Unternehmen nicht von seinen Mitbewerbern abheben müssen. Nicht-differenzierende Software stellt den Löwenanteil der Software, die ein Unternehmen nutzt, und jedem Unternehmen ist anzuraten, für die Entwicklung solcher Software auf die Beteiligung an Open Source zu setzen.

Literatur

Delio, M. (2004), 'Linux: Fewer Bugs Than Rivals', *Wired.com* . http://www.wired.com/news/linux/0,1411,66022,00.html [21. Feb 2007].

Ghosh, R. A., Glott, R., Krieger, B. und Robles, G. (2002), FLOSS Final Report - Part 4: Survey of Developers, *in* 'Free/Libre and Open Source Software: Survey and Study', International Institute of Infonomics, University of Maastricht and Berlecon Research GmbH. http://www.infonomics.nl/FLOSS/report/FLOSS_Final4.pdf.

IDC Software Consulting (2004), 'The Linux Marketplace - Moving from Niche to Mainstream', *Prepared for Open Source Development Labs (OSDL)* . http://old.linux-foundation.org/docs/linux_market_overview.pdf [21. Feb 2007].

Kerner, S. M. (2006), 'Coverity Study Ranks LAMP Code Quality', *Internetnews.com* . http://www.internetnews.com/stats/article.php/3589361 [21. Feb 2007].

Krass, P. (2002), 'A terrible thing to waste', *CFO IT* . http://www.cfoeurope.com/displayStory.cfm/1739037 [21. Feb 2007].

Microsoft Corp. (2006), 'Form Q-10 For the Quarter Ended March 31, 2006', *Prepared for United States Securities and Exchange Commission* . http://www.microsoft.com/msft/download/FY06/MSFT_3Q2006_10Q.doc [21. Feb 2007].

Raymond, E. S. (1999), *The Cathedral and the Bazaar: Musings on Linux and Open Source from an Accidental Revolutionary*, O'Reilly & Associates, Cambridge.

U.S. Bureau of Labor Statistics (2006), 'Occupational Outlook Handbook (OOH), 2006-07 Edition', online handbook. http://www.bls.gov/oco/ [21. Feb 2007].

U.S. General Accountion Office (1997), 'Immature Software Acquisition Processes Increase FAA System Acquisition Risks', *Prepared for Chairman, Subcommittee on Transportation, Committee on Appropriations, House of Representatives* . http://ntl.bts.gov/lib/000/200/234/ai97047.pdf [21. Feb 2007].

U.S. Office of Technology and Electronic Commerce (2003), 'Size of the U.S. Computer Software Industry', online report. http://web.ita.doc.gov/ITI%5CitiHome.nsf/AutonomyView/87200518f179196c85256cc40077ede1 [21. Feb 2007].