Westfälische Wilhelms-Universität Münster

MICE Economic Research Studies

Vol. 4

Open Source-Software
Eine volkswirtschaftliche Bewertung

Dr. Stefan Kooths
Dr. Markus Langenfurth
Dipl.-Volkswirtin Nadine Kalwey

Münster

Dezember 2003

MICE

Muenster Institute for Computational Economics

University of Muenster

mice.uni-muenster.de

Herausgeber:

Prof. Dr. Gustav Dieckheuer / Dr. Stefan Kooths

Kooths, Stefan / Langenfurth, Markus / Kalwey, Nadine:
Open Source-Software – Eine volkswirtschaftliche Bewertung
http://mice.uni-muenster.de/mers/mers4-OpenSource_de.pdf

MICE Economic Research Studies

ISSN 1612-9032

Volume 4

© MICE 2003

Westfälische Wilhelms-Universität Münster

Muenster Institute for Computational Economics

Fliednerstrasse 21

D-48149 Muenster/Germany

http://mice.uni-muenster.de

All Rights Reserved

Printed in Germany

Management Summary

Open Source-Software: Eine volkswirtschaftliche Bewertung

1. Im Kern kein Markt

- Die Open Source-Entwicklung

Die Entwicklung von Open Source-Software erfolgt bewusst außerhalb von Marktmechanismen, denn durch die freie Verfügbarkeit des Quellcodes soll gerade das Entstehen eines preisgesteuerten Softwaremarktes verhindert werden. Kommerzielle Geschäftsmodelle, die auf Open Source aufbauen, ändern daran insbesondere im Bereich der Standardsoftware nichts. Diesbezügliche Transaktionen vollziehen sich auf Komplementärmärkten und wirken allenfalls indirekt auf die Open Source-Softwareentwicklung zurück. In einer arbeitsteiligen Ökonomie erfüllt der Markt jedoch wichtige Informations-, Koordinations- und Anreizfunktionen. So werden über den Markt Kundenwünsche und Produktangebot zum Ausgleich gebracht (Konsumentensouveränität), knappe Ressourcen in die beste Verwendung gelenkt (optimaler Faktoreinsatz), Einkommen geschaffen und leistungsbezogen verteilt (produktivitätsorientierte Faktorentlohnung) sowie Innovationsanreize gesetzt (Fortschrittsfunktion). Preise sind dabei das zentrale Informationsmedium für Anbieter und Nachfrager, ohne das Märkte die genannten Funktionen nicht erfüllen können. Unentgeltlich („kostenlos") abgegebener Software fehlt aber gerade diese wichtige Eigenschaft, woraus sich erhebliche ökonomische Funktionsdefizite des Open Source-Modells ergeben.

2. „Happy Engineering"

– Entwicklerorientierung ist nicht Kundenorientierung

Für den marktfreien Kern des Open Source-Modells (geprägt durch restriktive Lizenzen wie die GPL) gilt, dass sich Entwickler hauptsächlich an Projekten beteiligen, die ihren persönlichen Präferenzen entgegen kommen, wobei individuelles Interesse an einer Problemlösung, die technische Herausforderung oder der Aufbau von Reputation wichtige Motivationskanäle darstellen. Damit sind in diesem Bereich die Entwicklerinteressen ausschlaggebend für Art und Umfang des Softwareangebots. Über diesen Anreizmechanismus entstehen vor allem anspruchsvolle Lösungen für versierte Nutzer. Ausschlaggebend für das Softwareangebot sollte jedoch nicht das technisch Mögliche, sondern das vom Nutzer tatsächlich Gewollte, Bedienbare und Bezahlbare sein. Es ist der Kunde, der als Souverän im

Marktprozess über seine Produktentscheidung letztlich den entscheidenden Einfluss auf das Angebot nimmt. Anbieter kommerzieller Software können auf einem funktionsfähigen Markt nur bestehen, wenn ihr Produktangebot den Kundenwünschen entspricht. Ihre Investitionen in Marktforschung dienen der Identifikation der Kundenwünsche und damit ihrem eigenen Bestehen im Marktprozess. Die Bewertung ihrer Produkte anhand der erzielbaren Absatzpreise und -mengen (Umsätze) einerseits und die Kalkulation der für die Produktentwicklung in Anspruch genommenen Ressourcen anhand von Faktorpreisen und -mengen (Kosten) andererseits generieren Gewinn- oder Verlustsignale, anhand derer sich Anbieter von Software in einem funktionsfähigen Markt orientieren können. Fehlt hingegen dieser Markt, so gibt es keinen verlässlichen Mechanismus, der die Entwicklerinteressen auf die tatsächlichen Kundenwünsche lenkt. Konsumentensouveränität ist ohne Produktpreise nicht durchsetzbar – der nicht selbst programmierende Softwarenutzer wird zum passiven Empfänger des Outputs der Open Source-Entwicklergemeinde. Dem Versuch, die Entwicklung von Open Source-Software zu professionalisieren, indem Programmierer für ihre Open Source-Beiträge von kommerziellen Anbietern entgolten werden, sind – wie in These 4 gezeigt – hingegen sehr enge Grenzen gesetzt. Die Entwicklerorientierung ist daher als dominierendes Produktionsprinzip des Open Source-Modells anzusehen.

3. Nicht kostenlos, aber manchmal umsonst – Die Open Source-Ressourcenlenkung

Wirtschaften bedeutet, die vorhandenen knappen Ressourcen so zu lenken, dass sie in die nach Einschätzung der Kunden beste Verwendung fließen. Ist die Software unentgeltlich, so kann auch die in der Entwicklung eingesetzte Arbeit nicht direkt monetär entgolten werden. Die mangelnde Bepreisung des Endproduktes überträgt sich unmittelbar auf die vorgeschalteten Faktormärkte und setzt dort marktwirtschaftliche Lenkungsmechanismen außer Kraft. Es können daher viele Entwickler an Programmen arbeiten, die niemand haben will oder Programme nicht entstehen, die von den Nutzern gewünscht werden. Selbst die weite Verbreitung eines Open Source-Produktes ist kein zuverlässiger Indikator für eine geglückte Ressourcenlenkung, da sie keinen Anhaltspunkt darüber gibt, ob der Einsatz der Entwicklungskapazität für ein alternatives Produkt zu einer höheren Befriedigung der Kundenbedürfnisse beigetragen hätte (mangelnde Transparenz der Opportunitätskosten). Die praktisch kostenlose Kopierbarkeit

von Software wird in der bisherigen Open Source-Diskussion in der Weise interpretiert, dass es unter den Nutzern keine Rivalität um Softwareprodukte gäbe, woraus dann eine Kollektivguteigenschaft und ein partielles Marktversagen abgeleitet wird. Diese Sichtweise greift aber zu kurz, weil sie nur auf bestehende (wie auch immer produzierte) Software Bezug nimmt (mangelnde Ex-post Rivalität). Hingegen herrscht vor der Entwicklung neuer Software Wettbewerb um den Einsatz knapper Entwicklungskapazität und damit um die Frage, welche neuen Produkte entwickelt werden sollen und welche nicht (Ex-ante Rivalität). Gerade im Softwaresektor mit seinen hochqualifizierten Beschäftigten sollte die Ressourcenlenkung nach marktwirtschaftlichen, d. h. produktivitätsorientierten Kriterien erfolgen, um die volkswirtschaftlich knappe Ressource „Entwicklungskapazität" nicht zu verschwenden.

4. Second Best Lösung

– Kommerzielle Open Source-Geschäftsmodelle

Da die kommerzielle Verwertung von Open Source-Software selbst eingeschränkt ist, nutzen kommerzielle Geschäftsmodelle Open Source, um den Verkauf komplementärer Dienste und Produkte zu fördern. Dabei sind kommerzielle Geschäftsmodelle darauf angewiesen, das komplementäre Produkt möglichst exklusiv vermarkten zu können. Ihre Anreize, in Open Source-Software zu investieren, richten sich nicht direkt auf das Open Source-Produkt, sondern indirekt auf den Nutzen, den Open Source-Software für ihr verwertbares Produkt hat. Kommerzielle Geschäftsmodelle müssen alle Investitionen in die Open Source-Entwicklung indirekt über die Erlöse bei komplementären Produkten finanzieren, sofern es sich nicht um reine Individualsoftware handelt. Diese Form der Quersubventionierung verfälscht das marktwirtschaftliche Preisgefüge, was zu erheblichen Verzerrungen in der Produktionsstruktur führen kann. So würde z. B. die Finanzierung von Softwareentwicklung über das Servicegeschäft die Servicepreise über die dort tatsächlich anfallenden Kosten treiben, wodurch die Nachfrage nach diesen Leistungen geringer ausfällt, als es in einer marktwirtschaftlichen Ordnung möglich wäre. Ökonomisch unterschiedliche Aktivitäten (Softwareentwicklung, Serviceleistungen) sollten auch unabhängig voneinander bepreist werden, um dem Nachfrager klar zu signalisieren, welche Ressourcen zur Befriedigung seiner Bedürfnisse tatsächlich in Anspruch genommen werden.

5. Die Schwächung kommerzieller Software stärkt nicht Open Source-Software

Als grundsätzliche Alternative für die Softwareentwicklung ist das Open Source-Modell ungeeignet. Man kann Open Source nicht stärken, indem man den kommerziellen Softwaremarkt schwächt, denn die Open Source-Softwareentwicklung bedingt einen starken kommerziellen Softwaremarkt. Dieser Markt dient der Open Source-Entwicklung als Ressourcenquelle für Arbeitsplätze, Einkommen und innovative Produktideen.

6. Open Source-Software fördert nicht den IT-Mittelstand

Open Source-Software bietet keine neuen, sondern nur einen Teil der bereits auf dem kommerziellen Softwaremarkt gegebenen Geschäftsmöglichkeiten. Als standortpolitische Maßnahme zur Förderung des IT-Mittelstandes ist die Förderung von Open Source ungeeignet. Mit Ausnahme der Individualsoftware liefert der marktfreie Koordinationsmechanismus in der Entwicklung im Vergleich zum kommerziellen Softwaremarkt keinen Wertschöpfungsbeitrag. Ist die Software unentgeltlich verfügbar, so entstehen mit ihrer Entwicklung anders als auf dem kommerziellen Softwaremarkt auch keine Erlöse, Einkommen, Arbeitsplätze und Steuern. Allein im Rahmen des Microsoft-Partnermodells erstellen unabhängige Softwarehäuser mit insgesamt 45.000 Beschäftigten Microsoft-bezogene Softwareprodukte im Wert von 6,6 Mrd. Euro. Geschäftsmodelle, die auf Open Source aufbauen und komplementäre Dienste und Produkte anbieten, unterscheiden sich in diesem komplementären Teil grundsätzlich nicht von kommerziellen Angeboten. Allerdings führt die Quersuventionierung des marktfreien Kerns von Open Source-Software dazu, dass die dort realisierte Wertschöpfung niedriger ausfällt als es bei kommerzieller Softwareproduktion. Insgesamt ist damit die Wertschöpfung des Open Source-Modells geringer.

7. Die Open Source-Förderung – Kein Instrument der Wettbewerbspolitik

Aus der Position einzelner Anbieter auf bestimmten Teilmärkten des IT-Sektors (insbesondere bei Desktop-Betriebssystemen und Büro-Software) lässt sich keine Rechtfertigung für die Förderung von Open Source-Software ableiten. Die staatliche Begünstigung/Subventionierung von Wettbewerbern in hochkonzentrierten Märkten ist kein Instrument der Wettbewerbspolitik, weil derartige Interventionen einen Informationsstand voraussetzen, über den staatliche Instanzen nicht verfügen können (z. B. zukünftige Marktentwicklung, optimale

Marktstruktur). In seiner Rolle als IT-Nachfrager sollte sich der Staat daher – wie in den Haushaltsgesetzen vorgesehen – strikt von Wirtschaftlichkeitsüberlegungen (TCO-Analysen) lenken lassen, auf die Beeinflussung der Marktentwicklung bewusst verzichten und den Schutz der Wettbewerbsordnung den Kartellämtern überlassen. Setzt man sich aber – aus welchen Gründen auch immer – über diese ordnungspolitischen Grundregeln hinweg, so bleibt zu fragen, weshalb ausgerechnet ein Produktionsverfahren unterstützt werden soll, das mit den oben genannten Koordinationsdefiziten verbunden ist.

Inhaltsverzeichnis

Abbildungs- und Tabellenverzeichnis

Einleitung

Nachdem sich die Erwartungen des Internet-Booms und die damit verbundenen Vorstellungen einer New Economy nicht erfüllt haben, richten sich die Hoffnungen nun auf ein neues Schlagwort aus der IT-Welt: Open Source-Software. Wieder soll eine neue und nach anderen Regeln funktionierende Ökonomie entstehen, diesmal auf der Basis einer Software, deren wichtigste ökonomische Eigenschaften die Abwesenheit von Eigentumsrechten und – daraus folgend – die unentgeltliche Verfügbarkeit sind. Der Tausch „Ware gegen Geld" soll dabei abgelöst werden durch einen Tausch „Geschenk gegen Reputation" und die Partizipation an einer Entwickler-Community auf Gegenseitigkeit. Das erklärte Ziel besteht darin, Software ohne Lizenzgebühren allgemein zugänglich zu machen, wodurch der über das Preissystem gesteuerte marktliche Koordinationsmechanismus außer Kraft gesetzt wird. Dabei sind die in der Öffentlichkeit mit Open Source-Software verbundenen Hoffnungen vielfältig: mehr Transparenz, mehr Demokratie, mehr Arbeitsplätze insbesondere bei kleinen und mittleren Unternehmen und nicht zuletzt ein Sanierungsbeitrag zu den Staatsfinanzen durch vermeintlich kostenlose Software.

Mit der Befürwortung einer Softwareproduktion, die im Kern auf Preise und damit den Marktprozess verzichtet, ist häufig eine grundsätzliche Skepsis gegenüber einem marktlichen Koordinationsmechanismus verbunden. Die Entwicklung von Standardsoftware auf Open Source-Basis erfolgt bewusst marktfrei und scheint so eine Herstellungsmethode zu bieten, die nicht den in einer Marktwirtschaft üblichen Prozessen entspricht. Das Fehlen eines Preises und die unentgeltliche Weitergabe der Software sind Ausdruck dieser Abwesenheit von Marktprozessen. Es ist das Ziel der Open Source-Lizenzen, eine Aneignung von Eigentumsrechten an der Software und so die Durchsetzung von Preisen zu verhindern. Dies hat zur Folge, dass mit der entwickelten Software auch keine Erlöse erzielt werden können.

Dennoch gibt es Geschäftsmodelle, die auf Open Source-Software aufbauen. Diese Geschäftsmodelle müssen jedoch grundsätzlich alle Investitionen in die Softwareentwicklung über komplementäre Leistungen finanzieren. Eine gewisse Ausnahme stellt hierbei die Auftragsentwicklung von Individualsoftware dar. Da die hierbei entstehende Software auf die speziellen Interessen des Auftraggebers zugeschnitten wird, findet eine anschließende Mehrfachverwertung nicht statt, so dass die (komplementäre) Leistung der Softwareentwicklung praktisch dem Verkauf einer

Einmallizenz gleichkommt. Zwar steht die erzeugte Software anschließend jedermann frei zur Verfügung, aufgrund der kundenspezifischen Eigenschaften ist sie aber nicht von allgemeinem Interesse und wird daher auch nicht von anderen Nutzern als dem Auftraggeber eingesetzt. Die von Open Source-Software erwarteten Vorteile werden damit aber bei Individualsoftware gerade nicht wirksam.

Komplementären Geschäftsmodellen außerhalb der individuellen Auftragsentwicklung sind hingegen sehr enge Grenzen gesetzt. So stehen z. B. die Ergebnisse von Investitionen in eine Open Source-Entwicklung auch gleichzeitig allen anderen Anbietern zur Verfügung. Diese können dann die Entwicklungsinvestitionen eines Wettbewerbers nutzen und eigene komplementäre Leistungen anbieten, ohne selbst Investitionen finanzieren zu müssen. Dies begrenzt wiederum insgesamt die Bereitschaft, in die Softwareentwicklung zu investieren. Im Gegensatz zur Erstellung von Individualsoftware stellt daher der Bereich der Standardsoftware kein ökonomisch tragfähiges Fundament für Open Source-Geschäftsmodelle dar. Auch wenn es somit Geschäftsmodelle gibt, die Open Source-Software nutzen, bleibt die Softwareentwicklung als der Kern des Open Source-Modells grundsätzlich marktfrei.

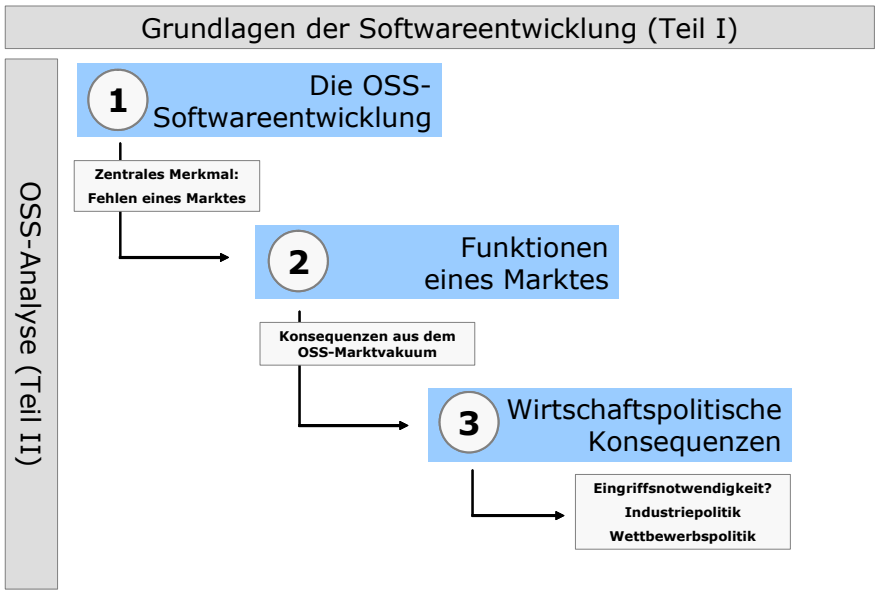
Preisen und dem Markt als Koordinationsinstrument kommen in einer Marktwirtschaft jedoch eine zentrale Bedeutung zu: So erfolgt auf Märkten ein Abgleich der Produktionspläne der Anbieter mit den Nachfragewünschen der Konsumenten, knappe Ressourcen werden in die produktivste Verwendung gelenkt und durch Property Rights an den Ergebnissen der eigenen Leistung werden Innovationsanreize gesetzt. Preise übernehmen dabei die zentrale Lenkungsfunktion. Sie zeigen die relativen Knappheiten an und sind der Wegweiser für Anbieter, durch den sie erkennen können, wie die Nachfrager den Wert einer Ware einschätzen und welche Waren am stärksten nachgefragt werden. Erst Preise ermöglichen eine arbeitsteilige Wirtschaft, ohne sie gibt es weder Umsätze noch Einkommen. Kein anderer Maßstab ist auch nur annähernd in der Lage, Kundenwünsche, Produktionsmöglichkeiten und neue Marktchancen gleichermaßen sinnvoll zu bewerten. Verzichtet die Open Source-Softwareentwicklung auf die Bepreisung ihrer Softwareprodukte, so kann der Markt seine zentralen Lenkungsfunktionen im Open Source-Modell nicht erfüllen. Keinen Preis zu haben, bedeutet deshalb volkswirtschaftlich weitaus mehr, als in den Augen der Nachfrager gratis zu sein.

Diese Studie untersucht, inwieweit das Open Source-Modell in der Lage ist, eine Alternative zur kommerziellen Softwareproduktion darzustellen. Hierzu wird die grundlegende Funktionsweise des Open Source-Modells untersucht und an den

Marktfunktionen gespiegelt, um herauszuarbeiten, welche Konsequenzen das Fehlen von Marktprozessen für diese Form der Softwareproduktion hat.

Im Mittelpunkt der Studie steht nicht die Geeignetheit von Open Source als Massnahme zur Erreichung anderer wirtschaftspolitischer Ziele. Lediglich im abschließenden Kapitel werden mögliche Konsequenzen der Untersuchungsergebnisse für verschiedene Politikbereiche aufgezeigt. Diese Studie macht somit keine Aussagen über die wettbewerbstheoretische Beurteilung von Marktpositionen auf dem Softwaremarkt. Würde man die Marktposition untersuchen wollen, so wäre eine grundsätzlich andere Vorgehensweise erforderlich. So wäre zunächst eine Abgrenzung des Marktes und Feststellung der Marktposition notwenig. Im Anschluss könnten verschiedene Massnahmen als Reaktion auf eine evtl. festgestellte marktbeherrschende Stellung überprüft werden. Gleichwohl kann an dieser Stelle darauf hingewiesen werden, dass die Förderung einer Alternativtechnologie in der Regel nicht zu den wettbewerbspolitischen Massnahmen zählt.

Abbildung 1: Aufbau der Studie



Die Studie ist in zwei Teile unterteilt. Der erste Teil legt allgemeine Grundlagen zum Softwaremarkt und zur Softwareentwicklung. Der zweite Teil stellt daraufhin kurz die Entstehungsgeschichte von Open Source-Software dar, um im Anschluss daran eine volkswirtschaftliche Bewertung von Open Source-Software durchzuführen und daraus wirtschaftspolitische Konsequenzen abzuleiten.

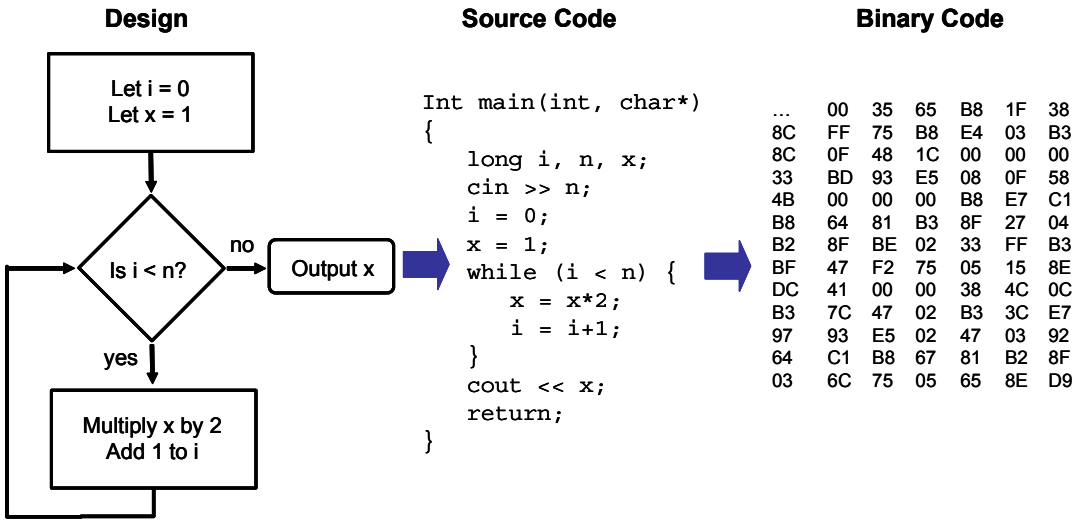
1. Grundlagen des Softwaremarktes

1.1 Merkmale von Software

1.1.1 Schritte in der Softwareentwicklung

Vereinfacht gesprochen besteht Software aus einer Liste von Anweisungen zur Verarbeitung von Daten. Software ist ein geistiges Werk, welches die Programme, Prozeduren, Richtlinien und die Dokumentation umfasst, die notwendig sind, um einem Computer Anweisungen zu erteilen und Aufgaben erfüllen zu lassen. Sie beinhaltet die Gesamtheit der formalisierten Arbeitsanweisungen und Funktionen, die zum Betrieb eines Computers benötigt werden (Betriebssystem, Programmiersprachen und deren Compiler oder Interpreter, Programme).[1]

Abbildung 2: Schritte in der Softwareproduktion



Quelle: EVANS UND REDDY (2002), S. 5.

Die Entwicklung von Software erfolgt idealtypisch in drei Schritten. Auf der Basis des speziellen Bedürfnisses, für das eine Software entwickelt werden soll, wird zunächst das Design als Bauplan für die spätere Programmierung festgelegt. Anschließend erfolgt die eigentliche Programmierung. Der Programmierer verfasst auf der Basis des Bauplans die Anweisungen in einer der Programmiersprachen. Diese Anweisungen bilden den Quellcode oder Source Code einer Software. Beim Quellco-

[1] Vgl. JANKO, BERNROIDER UND EBNER (2000), S. 13.

de handelt es sich um die Befehlsabfolge einer Software und damit um den Schlüssel zu ihrer Funktionsweise.

Höhere Programmiersprachen (wie PASCAL, C oder C++) sind in ihrer Darstellung an die – meist englische – Schriftsprache angelehnt. So sind z. B. „if" oder „when" übliche Befehle in zahlreichen Programmiersprachen. Die Übersetzung in den binären Code stellt schließlich den letzten Schritt in der Softwareentwicklung dar. Der Quellcode wird dabei durch Compiler oder Interpreter in eine für den Computer nutzbare Form übersetzt, der Quellcode wird zum Objektcode oder binären Code (bestehend aus Nullen und Einsen bzw. An/Aus-Anweisungen für die Schaltkreise des Prozessors). Abbildung 2 zeigt schematisch die einzelnen Schritte in der Softwareentwicklung.[2]

Innerhalb der Softwareentwicklung lassen sich folgende Trends identifizieren:[3]

- Steigende Bedeutung von Software- gegenüber Hardware-Produkten
- Steigende Bedeutung von softwarebezogenen Dienstleistungen gegenüber Software-Produkten
- Steigende Komplexität der Softwareprojekte
- Steigende Qualitätsanforderungen
- Entwicklung von der Individualsoftware zur Standardsoftware
- Zunehmende Anzahl neuer Versionen, die auf bereits existierender Software aufbauen.

Je weniger diese einzelnen Schritte der Softwareentwicklung für den allgemeinen Zugriff geöffnet sind, desto wirksamer ist das in die Softwareentwicklung eingeflossene Wissen geschützt und desto schwieriger ist die Programmierung einer eigenen auf diesem Wissen basierenden Software. Liegt nur das Softwaredesign als Bauplan vor, so ist die Struktur und Logik einer Software bekannt, die Programmierarbeit muss jedoch noch eigenständig geleistet werden. Besteht ein Zugriff auf den Quellcode, können der Ablauf und die einzelnen Programmierschritte nachvollzogen werden. Auch ist es möglich, die Software zu verändern oder einzelne Komponenten zu kopieren. Liegt nur der für Menschen nicht verständliche binäre Code vor, so sind

[2] Vgl. EVANS UND REDDY (2002), S. 5. Der binären Code in Abbildung 2 ist in hexadezimaler Schreibweise – einer einfachen Form der Darstellung von Nullen und Einsen – wiedergegeben.

[3] Vgl. BALZERT (1996), S. 27 und BERLECON RESEARCH (2002c), S. 24-25.

keine Rückschlüsse auf die Struktur und Programmierung der Software mehr möglich.[4]

Um die mit der Programmierung verbundenen geistigen Eigentumsrechte zu schützen, wird kommerzielle Software häufig in binärer Form vertrieben. Zwar besteht auch rein rechtlich durch ein Copyright oder Patent ein Schutz des geistigen Eigentums, eine Durchsetzung dieser Rechte ist jedoch nicht immer möglich.[5] Eine Rückübersetzung des binären Codes in den Quelltext – das sog. Reverse Engineering – ist hingegen nur unter großem Aufwand möglich. Einzelne Bestandteile der Software oder die Software insgesamt können auch durch Patente oder ein Copyright geschützt werden.[6] Hierdurch ist es ohne Zustimmung des Urhebers nicht mehr erlaubt, die Software zu kopieren oder - sofern Zugriff auf den Quellcode besteht - einzelne Bestandteile der Software in anderen Programmen zu verwenden. [7]

1.1.2 Entwicklung der Marktstruktur

Erste Ideen, mittels einer Speicherprogrammierung Universalrechnern bestimmte Aufgaben zuzuweisen, entstanden vor ca. 50 Jahren. Die Softwareindustrie entwickelte sich dann über verschiedene Stufen von einer vertikal integrierten zu einer horizontalen desintegrierten Marktstruktur.[8]

Bis Ende der 50er Jahre war der Softwaremarkt nur durch Individualsoftwareprojekte geprägt. Hardwarehersteller oder unabhängige Softwareunternehmen übernehmen dabei Aufträge, insb. der amerikanischen Regierung. Die weitere Verbreitung von Großrechnern führt dazu, dass der Bedarf an Software nicht mehr nur durch die EDV-Abteilungen von Unternehmen oder durch Hardwarehersteller gedeckt werden kann, sondern zunehmend auch von unabhängigen Softwareproduzenten übernommen wird. So entstehen in den 60er Jahren dann die ersten Softwarepakete, die mehrmalig verkauft werden können. Allerdings sind diese auf spezielle Bedürfnisse von Großunternehmen ausgerichtet.

---

[4] Vgl. BERLECON RESEARCH (2002c), S. 11, GRASSMUCK (2002), S. 233-234 und GRÖHN (1999), S. 4, dort zitiert.

[5] Vgl. SCHMIDT UND SCHNITZER (2003), S. 4.

[6] So ist z. B. der Algorithmus zur Erstellung von mp3-Dateien von der Frauenhofer Gesellschaft patentiert. Jede Software, die diesen Algorithmus zur mp3-Erstellung verwendet, muss von der Frauenhofer Gesellschaft lizenziert sein.

[7] Vgl. EVANS UND REDDY (2002), S. 6.

[8] Vgl. zur Entwicklungsgeschichte HOCH, U. A. (2000) 259ff und JANKO, BERNROIDER UND EBNER (2000), S. 16-17. Siehe auch www.softwarehistory.com.

Die 70er Jahre bis zu Beginn der 80er Jahre sind von einer Desintegration von Hard- und Softwareerstellung gekennzeichnet. Die Entscheidung von IBM, Hard- und Software getrennt zu vermarkten („unbundling" policy), führt dazu, dass ein reichhaltiges Angebot an betriebswirtschaftlicher Standardsoftware für verschiedene Branchenanwendungen entsteht. Anbieter von betrieblicher Software wie SAP (1972) oder BaaN (1978) haben ihren Ursprung in dieser Zeit.

Während in der Periode bis Anfang der 80er Jahre Computer nur als Großrechner verbreitet waren und dementsprechend auch Software nur für diese Bedürfnisse entwickelt wurde, wird der Computer mit Vorstellung des IBM Personal Computers am 12. August 1981 zu einem Massenprodukt.[9] Da IBM auch hier die Hardware getrennt von der Software vermarktet, entsteht ein eigenständiger Markt für Endbenutzersoftware.[10]

Neben dem Softwarebereich, der sich aus dem Hardwarebereich herauslöste, hat sich auch ein eigenständiger Dienstleistungsbereich entwickelt. EDV-Dienstleister übernehmen Beratung, Implementierung, Schulungen oder Rechenzentrumsleistungen sowie die Erstellung von Individualsoftware. Die Unterscheidung zwischen Softwareentwicklern und Softwaredienstleistern ist dabei allerdings teilweise auch künstlich und im Einzelfall schwierig zu ziehen. So verfügen große Hersteller von betrieblicher Software auch häufig über Serviceabteilungen. Ebenso können IT-Dienstleistungen Entwicklungskomponenten enthalten.[11]

Aus dieser Entwicklungsgeschichte ergibt sich die heute festzustellende Struktur des IT-Marktes. Der folgende Abschnitt untersucht aus ökonomischer Sicht die Besonderheiten des Softwaremarktes.

1.2 Eine ökonomische Analyse des Softwaremarktes

1.2.1 Ökonomische Merkmale von Software

Untersucht man Software anhand ökonomischer Kriterien, lassen sich besondere Merkmale identifizieren. Diese Softwaremerkmale haben Konsequenzen für den Entwicklungsprozess und den Wettbewerb auf dem Softwaremarkt. Im Folgenden

[9] Mit dem Altair 8800 im Jahre 1975 und dem Apple II im Jahre 1977 hat es bereits vorher Personal Computer gegeben. Sie konnten sich jedoch nicht als längerfristige Standardplattform durchsetzen.

[10] Vgl. LEHRER (2000), S. 590.

[11] Vgl. LEHRER (2000), S. 589.

werden die aus ökonomischer Sicht relevanten Besonderheiten von Software aus angebots- und nachfrageseitiger Sicht dargestellt:[12]

1.2.1.1 Angebotsseitige Merkmale

- **Sehr hohe Entwicklungskosten („first-copy-costs")**

  Software ist ein geistiges Werk oder „Product of intellectual property rights". Damit fallen die größten Herstellungskosten als versunkene Kosten für die Entwicklung und das Testen der Software vor dem Vertriebsstart an.[13] Hieraus kann sich die Notwendigkeit ergeben, neu entwickelte Software möglichst schnell kommerziell zu verwerten.

- **Sehr geringe Grenzkosten der Produktion**

  Sofern es sich um Standardsoftware handelt, liegt die fertige Software in digitaler Form vor und kann beliebig oft kopiert werden. Die Kopierkosten und damit die Grenzkosten der Produktion sind sehr gering. Die Softwareindustrie ist deshalb von degressiven Durchschnittskostenverläufen geprägt.[14] Damit bestehen in der Softwareproduktion auch kaum Produktionsbeschränkungen. Sofern Software nicht rein elektronisch vertrieben wird, können sich angebotsseitige Engpässe nur bei der Vervielfältigung, bei der Verpackung und beim Vertrieb ergeben.

- **Verbundvorteile in der Produktion**

  Einmal geschriebene Programmelemente können in anderen Programmen eingesetzt werden. Daher spielen auch Verbundvorteile eine wichtige Rolle. Grundsätzlich ist es dabei möglich, dass über definierte Schnittstellen auch andere Produzenten an Verbundvorteilen partizipieren können. Dadurch können einzelne Teile oder Komponenten einer Software auch anderen Softwareproduzenten zur Verfügung gestellt werden.[15] So ermöglichen die sog. APIs (Application Programming Interfaces) innerhalb des Betriebssystems Windows ein Aufrufen einzelner Komponenten des Betriebssystems.

[12] Siehe zu den Merkmalen auch BALZERT (1996).

[13] Vgl. OECD (2002), S. 105.

[14] Vgl. OECD (2002), S. 105 und GRÖHN (1999), S. 5.

[15] Vgl. GRÖHN (1999), S. 5.

- **Netzwerkeffekte auf der Angebotsseite**

  Je mehr Anwendungen es für ein bestimmtes Betriebssystem gibt, desto attraktiver ist dieses für den Nutzer. Mit der Nutzerzahl wiederum steigt die Attraktivität für Entwickler, neue Software für eine bestimmte Plattform zu entwickeln. Anders als Skalenerträge in der Produktion von Software sind Netzwerkeffekte nicht auf ein Unternehmen beschränkt. Andere Anbieter können für das gleiche Netzwerk produzieren.[16] Sie profitieren dabei von einem etablierten Standard, denn sie müssen ihre Software nicht für verschiedene Plattformen auslegen.

- **Immaterialität**

  Die Immaterialität von Software und der hohe Know-how-Anteil führen dazu, dass eine Messung des Wertes von Software nur schwer möglich ist. Auch lässt sich der Entwicklungsfortschritt von Software nur schwierig monetär quantifizieren. Dies führt dazu, dass produktspezifische Eigenschaften wie die Qualität oder ein Vergleich zwischen Produkten oder Standards ebenfalls erschwert werden.[17]

- **Internationalität**

  Die Eigenschaften von Software führen dazu, dass der Softwareproduktion und dem Vertrieb Internationalisierungstendenzen inhärent sind. Sowohl die Entwicklungszusammenarbeit wie auch der Vertrieb der fertigen Software können weltweit verteilt erfolgen. Die geringen Grenzkosten in der Produktion legen es nahe, bei einer entsprechenden Anpassung der Menüsprache eine fertig entwickelte Software weltweit zu vertreiben. Bei einem Vertrieb über das Internet ist ein fast zeitgleicher Weltmarkteintritt auch für kleinere Softwareanbieter möglich.

1.2.1.2 Nachfrageseitige Merkmale

- **Kein Verschleiß und kein Verfallsdatum – aber: Software veraltet**

  Software nutzt sich nicht ab und hat auch kein Verfallsdatum. Damit kann Software theoretisch beliebig lange eingesetzt werden. Begrenzt wird die Einsatzfähigkeit einer Software lediglich durch die Funktionsfähigkeit der zugehörigen Hardware. Mit der Weiterentwicklung der Hardware, insb. der Prozessortechnologie, ergibt sich die Möglichkeit zu einer Leistungssteigerung der Software, wodurch deren Funktionsumfang oder Einsatzbereich erweitert werden.

[16] Vgl. GRÖHN (1999), S. 3.

[17] Vgl. JANKO, BERNROIDER UND EBNER (2000), S. 14.

Möchte der Nutzer das gesamte Leistungsspektrum einer neuen Hardwaregeneration ausnutzen, ist er auf eine Aktualisierung seiner Software angewiesen. Software ist dann veraltet, wenn eine neue Version neue gewünschte Funktionen enthält oder bereits bestehende Funktionen besser ausgeführt werden können.

- **Netzwerkeffekte auf der Nachfrageseite**

  Je mehr Nutzer ein bestimmtes Betriebssystem oder eine Anwendungssoftware verwenden, desto einfacher ist es, Dateien auszutauschen oder aber auch Hilfestellungen bei der Programmbenutzung zu bekommen. Damit steigt der Nutzen aus der Verwendung eines bestimmten Programms mit der Anzahl der anderen Nutzer, die dieses Programm verwenden. Mit der Entscheidung für ein bestimmtes Betriebssystem oder eine Software tritt der Anwender somit dem Netzwerk der Nutzer dieser Software bei.

  Zwischen verschiedenen Netzwerken müssen aber keine unüberbrückbaren Grenzen bestehen. So kann etwa durch die Konvertierung von Formaten die Lesbarkeit von Daten auf verschiedenen Plattformen gewährleistet sein, wodurch ein Wechsel und auch eine Zusammenarbeit zwischen verschiedenen Netzwerken möglich werden.

- **Erfahrungsgut und Lock-in-Effekte**

  Software ist ein Erfahrungsgut. Die Qualität und Eignung einer bestimmten Software kann i. d. R. erst nach intensiver Benutzung und Einarbeitung erkannt werden. Zusammen mit den Netzwerkeffekten kann es so zu einem Lock-in-Effekt kommen. Ist man in eine bestimmte Software eingearbeitet und liegen alle Dateien in einem Format für diese Software vor, so ist der Wechsel zu einer anderen Software schwierig. Ein Wechsel zu einer anderen Software ist für den Nutzer nur dann sinnvoll, wenn der Zusatznutzen dieser Software die erneut erforderlichen Lernkosten übersteigt.

- **Nichtrivalität im Konsum**

  Eine bestimmte Kopie einer Software kann rein physikalisch beliebig oft installiert und von verschiedenen Nutzern verwendet werden. Es besteht eine Nichtrivalität im Konsum.

Aus der bisherigen Darstellung geht hervor, dass Netzwerkeffekte, die Nichtrivalität im Konsum und die Möglichkeiten zur Exkludierbarkeit auf dem Softwaremarkt von besonderer Bedeutung sind. Daher soll auf diese Aspekte in den folgenden beiden Abschnitten noch einmal detaillierter eingegangen werden.

1.2.2 Netzwerkeffekte

Bei Netzwerkeffekten handelt es sich um einen Spezialfall technologischer externer Effekte. Externe Effekte liegen dann vor, wenn jemand von Konsum- oder Produktionshandlungen durch Haushalte oder Unternehmen betroffen ist, ohne dafür zu bezahlen bzw. entschädigt zu werden.[18] Treten externe Effekte auf, so sind die Marktpreise verzerrt und geben die tatsächlichen Knappheits- und Nutzungsverhältnisse nicht wieder, denn für einen Teil des Nutzens oder der Schädigung erfolgt keine Kompensation über den Markt.[19] Die Höhe externer Effekte bestimmt sich als die Differenz zwischen den gesellschaftlichen Kosten und Nutzen und den über den Markt abgegoltenen privaten Kosten und Nutzen. Greift der Staat nicht ein, so kommt es bei Gütern, die einen positiven Zusatznutzen erzeugen, zu einer Unterversorgung und bei Gütern mit negativen externen Effekten zu einer Überproduktion.

Die auf dem Softwaremarkt auftretenden Netzwerkexternalitäten sind ein Sonderfall externer Effekte, sie zeichnen sich durch die gemeinsame Verwendung eines gleichen Gutes aus. Treten durch den Anschluss an ein Netz (hier die Verwendung der gleichen Software) bei den anderen Nutzern dieses Netzes Nutzenveränderungen auf, ohne dass diese über den Marktpreis berücksichtigt werden, so liegen Netzwerkexternalitäten vor.[20] Netzwerkexternalitäten können in technischen Netzen wie einem Telekommunikationsnetz durch eine physische Verbindung der Netzteilnehmer untereinander entstehen oder in virtuellen Netzen, deren Verbindung in der Verwendung eines einheitlichen Standards (z. B. ein gemeinsames PC-Betriebssystem) besteht, auftreten.[21]

Man unterscheidet zwischen direkten und indirekten Netzwerkeffekten. Der direkte Effekt besteht darin, dass die Verwendung eines Programms schon allein deshalb vorteilhaft ist, weil viele andere Nutzer dieses Programm auch verwenden. Der indirekte Effekt liegt in der Verfügbarkeit von Komplementärprodukten und -diensten. Der Wert eines Softwareprogramms ist damit abhängig von der Verfügbarkeit von komplementären Produkten und Dienstleistungen.

---

[18] Vgl. FRITSCH, WEIN UND EWERS (2003), S. 92ff. GROSSEKETTLER (1995), S. 510.

[19] „Extern" bedeutet somit „außerhalb des Preissystems als dem Hauptkoordinationsmechanismus einer Marktwirtschaft stehend". GROSSEKETTLER (1995), S. 510.

[20] Vgl. WEIZSÄCKER, VON UND KNIEPS (1989), S. 458, KATZ UND SHAPIRO (1994), S. 93-115.

[21] Vgl. BLANKART UND KNIEPS (1992), S. 73. Diese bezeichnen materielle Netze als Hardwarenetze und immaterielle Netze als Softwarenetze. Vgl. auch KLODT U. A. (1995), S. 40.

Ist der Nutzen eines Netzwerkes positiv von der Teilnehmerzahl in einem Netz abhängig, so ergibt sich die ökonomische Relevanz von Netzwerkexternalitäten aus dem Erreichen einer *kritischen Masse* an Netzteilnehmern.[22] Erst bei einer genügenden Anzahl von Teilnehmern lohnt sich ein Netzaufbau oder der Wechsel in ein alternatives Netz. Ergibt sich diese kritische Teilnehmeranzahl nicht, so kann es sein, dass sich ein Netz nicht herausbildet oder es zum Verharren im bestehenden Netz kommt. Dieser bei einem Netzwechsel relevante Lock-in-Effekt aufgrund des Nichterreichens der kritischen Masse kann als Markteintrittsbarriere für alternative Anbieter wirken, die ein technologisch zum etablierten Anbieter identisches Netz aufbauen oder sich auf den Verbleib in einem technologisch veralteten Netz beziehen. Darüber hinaus kann es zu einer Netzzersplitterung kommen, wenn die kritische Masse für mehrere gleichartige Netze zwar erreicht wird, diese aber aufgrund unterschiedlicher Technologien nicht kompatibel sind und daher Netzexternalitäten nicht voll ausgeschöpft werden. Der Aufbau eines zweiten Netzes ist daher volkswirtschaftlich nicht sinnvoll.

Das Problem des Lock-in-Effekts tritt vor allem dann auf, wenn mit einem Netzwechsel irreversible Kosten verbunden sind, unvollständige Informationen und asymmetrische Präferenzen bestehen. Für einen Netzwechsler mit starken Präferenzen für die neue Technologie ist dann keine Sicherheit gegeben, dass diesem auch weitere Nutzer folgen werden. Ein strategisches Verhalten und Abwarten auf die Reaktion anderer Nutzer könnte dann den Wechselprozess erst gar nicht beginnen lassen.

1.2.3 Nichtrivalität und Exkludierbarkeit

Der vorangehende Abschnitt hat auf die Bedeutung der Eigenschaften Nichtrivalität im Konsum und Exkludierbarkeit von Software hingewiesen. Im Rahmen der Kollektivgütertheorie sollen diese Eigenschaften jetzt noch genauer untersucht werden.

Die Theorie der Kollektivgüter ermöglicht es, aus dem Vorliegen von Gütereigenschaften Rückschlüsse auf die Funktionsfähigkeit eines Marktes zu ziehen. Im Mittelpunkt der Kollektivgütertheorie steht dabei nicht die Frage, ob ein Markt wettbewerblich oder monopolistisch organisiert ist, sondern die Identifikation von Funkti-

[22] Vgl. KLODT U. A. (1995), S. 40, und BLANKART UND KNIEPS (1992), S. 79.

onsdefekten, die das Entstehen von selbstorganisierten Märkten verhindern.[23] Liegt ein Kollektivgut vor, so kann der Markt seine Allokationsfunktion nicht erfüllen: Es kommt entweder gar kein Angebot zustande oder Nachfrage und Angebot sind nicht synchronisiert, was im Ergebnis zu einer Unterversorgung bzw. einer Übernutzung des Kollektivgutes führt.

Das Hauptunterscheidungsmerkmal zwischen Kollektiv- und Individualgütern ist die Möglichkeit, Kollektivgüter durch mehrere Nachfrager gleichzeitig zu nutzen. Während bei Individualgütern jeder Konsument seine Nachfrage einzeln artikuliert, muss daher bei Kollektivgütern die Nachfrage der Nutzergruppe erst organisiert werden, um so den Bedarf und die Finanzierung zu ermitteln.[24] Kommt es nicht zu einer solchen Nachfrageorganisation, so entsteht auch kein Markt für dieses Gut. Für verschiedene Kategorien von Kollektivgütern lassen sich Regeln für ihre Finanzierung und Bereitstellung ableiten. Zur Identifikation dieser Güterkategorien werden die Kriterien *privatrechtliche Exkludierbarkeit* und *Rivalität im Konsum* herangezogen.[25] Während der Exklusionsgrad ε anzeigt, ob Nachfrager von der Nutzung des Gutes ausgeschlossen werden können und so ein selbstorganisiertes Angebot über den Markt zustande kommen kann, gibt der Rivalitätsgrad λ Auskunft über die Kosten eines zusätzlichen Nutzers und damit die volkwirtschaftlich sinnvolle Bepreisung eines Gutes.

Können einzelne Nachfrager nicht zu vertretbaren Kosten vom Gebrauch eines Gutes abgehalten werden, so liegt Nichtexkludierbarkeit ($\varepsilon = 0$) vor.[26] In diesem Fall sind die Eigentumsrechte an einem Gut nicht zugeordnet bzw. lassen sich nicht durchsetzen. Die Möglichkeit, Nutzer vom Konsum eines Gutes auszuschließen, ist entscheidend für das Zustandekommen eines Marktes. Ist ein wirksamer Ausschluss von Nachfragern nicht möglich, so entsteht für das betreffende Gut kein Markt, denn potenzielle Konsumenten werden – unter der Erwartung, einen eigenen Finanzierungsbeitrag leisten zu müssen – nicht bereit sein, ihren Bedarf offen zu artikulieren. Da der Konsument weiß, dass er nicht vom Konsum ausgeschlossen werden kann, könnte er das Gut auch als Trittbrettfahrer ohne eigenen Finanzierungsanteil

[23] Vgl. BURR (1995), S. 41, und GROSSEKETTLER (1991), S. 119-120.

[24] Vgl. GROSSEKETTLER (1995), S. 499.

[25] Vgl. GROSSEKETTLER (1995), S. 496.

[26] Wirtschaftlich vertretbar ist eine Exklusionstechnik, wenn sie Überfüllungs- und Verschwendungskosten reduziert, Trittbrettfahrerverhalten verhindert und der Wert der vermiedenen Kosten nicht kleiner ist als der Wert der Exklusionskosten.

nutzen. Es entsteht die Situation eines Gefangenendilemmas, in der jeder Einzelne rational handelt, sich aber gesamtwirtschaftlich ein ineffizientes Marktergebnis einstellt.[27] Ein Unternehmer wird nicht bereit sein, ein Gut mit Nichtexkludierbarkeitseigenschaft anzubieten, denn er hat keine privatrechtlichen Mittel, einen Finanzierungsbeitrag bei den Nutzern durchzusetzen – es kommt somit nicht zur Produktion dieses Gutes.[28]

Tabelle 1: Güterkategorien

	Rivalitätsgrad	
Exkludierbarkeit	**Nichtrivalität** $\lambda = 0$	**Rivalität** $\lambda = 1$
Nichtexkludierbarkeit $\varepsilon = 0$	Prototypische Kollektivgüter (z. B. Leuchtturm)	Quasikollektivgüter (z. B. Ozonschicht)
Exkludierbarkeit $\varepsilon = 1$	Klubkollektivgüter (z. B. Software)	Individualgüter (z. B. Lebensmittel)

Quelle: GROSSEKETTLER, H. (1995a), S. 499.

Für die Erstellung von Software ist das Merkmal der Exkludierbarkeit damit von wesentlicher Bedeutung. Besteht keine wirksame Möglichkeit der Exklusion und der Durchsetzung von Lizenzrechten, so haben Unternehmen aufgrund der beschriebenen Trittbrettfahrerproblematik keine Anreize, in die Softwareproduktion zu investieren und sie kommerziell zu vertreiben. Dieser Aspekt ist insbesondere für Open Source-Software von Bedeutung. Hier werden die Exklusion und damit die Durchsetzung von Eigentumsrechten von den verwendeten Lizenzen explizit ausgeschlossen.

Bei einem Rivalitätsgrad von null werden für die Versorgung weiterer Nachfrager keine zusätzlichen Einheiten dieses Gutes benötigt, die Grenzkosten der Versorgung betragen dann null, oder die Bereitstellung der zusätzlichen Menge verursacht keine Kosten.[29]

Kombiniert man die Extremwerte der Kriterien zur Einteilung von Kollektiv- und Individualgütern, so lassen sich vier Gruppen bilden (Tabelle 1).[30] Wenn eine Ex-

[27] Vgl. BURR (1995), S. 28.

[28] Vgl. GROSSEKETTLER (1995a), S. 496.

[29] Vgl. GROSSEKETTLER (1995a), S. 502-504.

[30] Vgl. zur Einteilung von Güterarten GROSSEKETTLER (1995a), S. 500-501.

kludierbarkeit nicht möglich ist und ein zusätzlicher Nutzer ohne zusätzliche Kosten versorgt werden kann, so liegt ein polares oder *prototypisches Kollektivgut* vor. Bei *Quasikollektivgütern* entsteht durch zusätzliche Nutzer ein Nutzenverlust, sobald an der Kapazitätsgrenze Überfüllungserscheinungen auftreten. In diesem Fall wäre aufgrund der mit einer Übernutzung entstehenden Kosten eine Exklusion weiterer Nachfrager notwendig, allerdings ist diese technisch nicht möglich oder ökonomisch ineffizient. Die Eigentumsrechte an Quasikollektivgütern sind nicht einzelnen Personen zurechenbar, sondern sind der Allgemeinheit oder einer bestimmten Gruppe zugeordnet.[31] Als besonderes Problem besteht bei Quasikollektivgütern die Gefahr einer Übernutzung und damit Zerstörung des Gutes.[32] Demgegenüber können bei *Klubkollektivgütern* Nutzungsberechtigte und Nicht-Nutzungsberechtigte leicht mit privatrechtlichen Mitteln voneinander getrennt werden (im Falle von Software mittels Lizenzen). Rivalität zwischen den Nutzungsberechtigten besteht jedoch nicht. Bei *Individualgütern* können Verfügungsrechte zugeordnet werden und es besteht Rivalität im Konsum.

In diesem Schema ist Software als Klubkollektivgut einzuordnen. Ein Ausschluss ist durch die explizite Definition von Verfügungsrechten möglich, eine Rivalität in der Nutzung besteht für Software nicht. Mit dieser Einordnung von Software ist die Empfehlung einer privatwirtschaftlichen Bereitstellung der Software verbunden. Für staatliche Eingriffe besteht keine Notwendigkeit.

1.2.4 Besonderheiten des Wettbewerbs auf dem Softwaremarkt

Aus den Merkmalen von Software ergeben sich spezielle Besonderheiten des Wettbewerbs auf Softwaremärkten. Diese sind im Folgenden dargestellt:

- **Wettbewerb als Innovationswettlauf und Wettbewerb um den Markt**

  Die zuvor dargestellten besonderen Merkmale von Software führen zu einer dem Softwaremarkt eigenen Innovationsdynamik. Geringe marginale Kosten und hohe Fixkosten für Forschung und Entwicklung können dazu führen, dass zur Finanzierung der Entwicklungskosten möglichst schnell möglichst viele Kopien verkauft werden müssen.

  Auch die auf dem Softwaremarkt bestehenden Netzwerkeffekte verstärken den Druck, Märkte möglichst schnell zu erschließen. Kann man eine bestimmte

[31] Vgl. BURR (1995), S. 30.

[32] Vgl. GROSSEKETTLER (1991), S. 70.

Technologie oder ein Dateiformat als Erster etablieren, ist es einfacher, den gesamten Markt zu gewinnen. Aufgrund dieser bestehenden „first-mover-advantages" kann es eher zu einem Wettbewerb um den Markt (das Netzwerk), als zu einem Wettbewerb im Markt kommen.

Ist ein Markt einmal besetzt, ist es schwierig, ein ähnliches sog. „Me-too-Produkt" mit geringer zusätzlicher Funktionalität zu entwickeln und im Markt zu platzieren, es sei denn, die Entwicklungskosten sind sehr gering. Bei einem „Me-too-Produkt" kann der Preiswettbewerb sehr schnell die Profitabilität zerstören. Daher versuchen Wettbewerber, ihre Produkte zu differenzieren oder weichen bereits besetzten Märkten aus.[33]

Die beschriebenen Faktoren fördern auf dem Softwaremarkt auch die Konzentrationstendenzen.[34] Kommt es auf dem Softwaremarkt somit zur Herausbildung einer dominanten Technologie oder Firma, so muss dies nicht das Ergebnis eines Marktversagens sein, sondern lässt sich auf die spezifischen Angebots- und Nachfragebedingungen auf dem Softwaremarkt zurückführen.[35]

- **Wettbewerb auch mit den eigenen Produkten**

  Da sich Software nicht verbraucht, gibt es auch für einen Anbieter auf einem gesättigten Markt die Notwendigkeit, sein Produkt beständig weiterzuentwickeln, um auf dem Markt bestehen zu können.[36] Es kommt nur dann zu einem Neukauf und Austausch der Software, wenn die neue Version spürbare Verbesserungen enthält. Will ein Unternehmen neue Versionen einer Software verkaufen, so besteht Wettbewerb damit auch darin, besser als die Vorgängerversion zu werden.[37] Damit kann es auch nicht zu einer dauerhaften Dominanz einer inferioren Technologie kommen, da diese entweder von Konkurrenzprodukten oder von neuen Versionen derselben Software verdrängt wird.

- **„Two-sided" Market**

  Der Markt für Betriebssysteme als ein Bestandteil des Softwaremarktes weist die Charakteristika eines „Two-sided" Market auf. Solche Märkte zeichnen sich

[33] Vgl. EVANS UND REDDY (2002), S. 16.

[34] Die Tendenz zur Konzentration ist im Desktop-Markt stärker als in dem Markt, in dem professionelle IT-Mitarbeiter Software einsetzen. Trainings- und Popularitätsbarrieren sind hier in der Regel nicht so hoch. IT-Mitarbeiter können leichter neue Programme erlernen und machen ihre Kaufentscheidung nicht von der Popularität abhängig. Vgl. HOCH U. A. (1999).

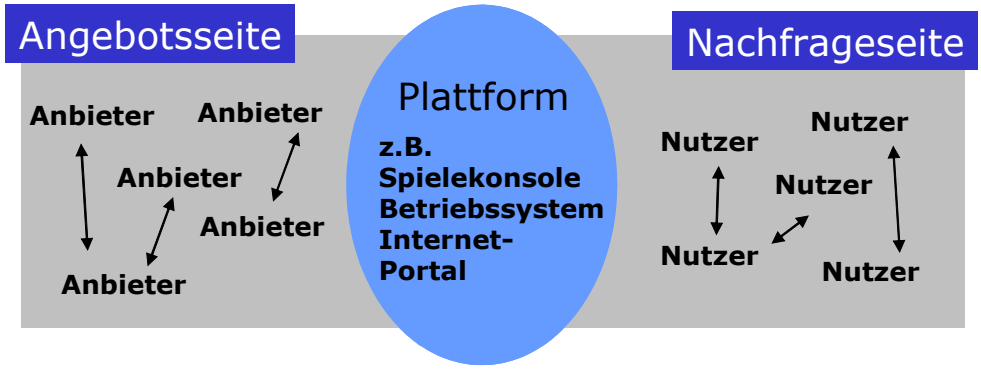
[35] Vgl. EVANS UND REDDY (2002), S. 17 und OECD (2002), S. 105.

[36] Vgl. GRÖHN (1999), S. 3.

[37] Vgl. SCHMIDT UND SCHNITZER (2003), S. 8.

durch Netzwerkeffekte auf der Angebots- und Nachfrageseite aus. Typische Plattformen mit den Merkmalen eines „Two-sided" Market sind Betriebssysteme, Spielekonsolen oder auch Internetportale. Der ökonomische Wert einer Plattform besteht darin, Anbieter und Nutzer zusammenzubringen. Die Abbildung 3 zeigt in einer vereinfachten Darstellung einen „Two-sided" Market.

Abbildung 3: „Two-sided" Market



Der Wert einer Plattform für den Nutzer steigt, je mehr Anbieter für eine Plattform produzieren oder ihre Dienste für diese Plattform anbieten. Gleichzeitig ist eine Plattform umso attraktiver für Anbieter, je mehr Nutzer sich für eine bestimmte Plattform entscheiden. Das Angebot auf einem „Two-sided" Market hängt somit von der Anzahl der Nachfrager ab, die eine bestimmte Plattform nutzen. Gleichzeitig nutzen umso mehr Nachfrager eine Plattform, desto mehr Anbieter auf ihr tätig sind. Die Entscheidung für eine bestimmte Plattform muss dabei nicht mit spezifischen Investitionen, etwa bei einer Spielekonsole, verbunden sein. So könnten Nutzer z. B. ohne großen Aufwand zwischen verschiedenen Internetseiten wechseln, auf denen Güter versteigert werden. Dennoch ist in der Realität zu beobachten, dass sich Anbieter und Nutzer auf die Plattformen konzentrieren, auf die die meisten anderen Anbieter und Nutzer zugreifen.

- **Geringe Markteintrittsschranken, ständige Bedrohung durch neue Anbieter**

  Weil sich auch andere Unternehmen die Netzwerkeffekte nutzbar machen können, indem sie für das gleiche Netzwerk produzieren wie der dominante An-

bieter, sind Märkte mit Netzwerkeffekten eher angreifbar als Märkte mit Skalenerträgen.[38]

Die Eintrittsbarrieren für neue Anbieter sind auf dem Softwaremarkt gering. Zwar fallen hohe Entwicklungskosten vor dem Markteintritt an, daneben sind jedoch im Vergleich zur Produktion physischer Güter nur geringe zusätzliche Investitionen erforderlich, um in den Markt eintreten zu können. Schnelle Innovationszyklen und technischer Fortschritt begünstigen ebenfalls die Markteintrittschancen für neue Anbieter und bedrohen die Dominanz etablierter Unternehmen.[39]

Die Höhe der Markteintrittsbarrieren variiert allerdings mit der Bedeutung der angebots- und nachfrageseitigen Netzwerkeffekte. Je stärker der Nutzen einer Plattform oder Technologie auch von diesen Netzwerkeffekten abhängt, desto größer muss der Zusatznutzen einer alternativen Plattform sein und desto schwieriger ist der Markteintritt.

- **Zerbrechliche Marktführer**

  Die vergleichsweise geringen Hürden für den Markteintritt fördern hohe Innovationsraten. Zahlreiche neue Softwaretechnologien sind in der Vergangenheit von kleinen innovativen Unternehmen in den Markt gebracht worden. Diese konnten dann aufgrund der Überlegenheit ihrer Technologie oder der innovativen Anwendung sehr schnell große Marktanteile erreichen und etablierte Anbieter ablösen. Sofern ein innovatives Produkt vorliegt sind keine weiteren Investitionen zum Markteintritt notwendig. Es ist daher nicht von der Größe eines Unternehmens abhängig, in Softwaremarkt erfolgreich mit einem neuen Produkt einzutreten. Entwickeln die im Markt vertretenen Anbieter ihre Technologie nicht weiter, so können neue Marktteilnehmer durchgängig die Marktposition der etablierten Anbieter bedrohen.[40]

  Markteintritte sind dabei auch in kleinen Teilmärkten oder mit Zusatzkomponenten zu bestehenden Softwarepaketen möglich. Da etablierte Anbieter mit ihren Angeboten zur Abdeckung einer möglichst breiten Nutzerzahl selten auf spezielle Bedürfnisse eingehen können, verbleiben Marktteile, die von alternativen Anbietern besetzt und ausgebaut werden können.

[38] Vgl. GRÖHN (1999), S. 3.

[39] Vgl. SCHMIDT UND SCHNITZER (2003), S. 7.

[40] Vgl. HOCH U. A. (1999) und SCHMIDT UND SCHNITZER (2003), S. 7.

- **Große Bedeutung von Standards und Consistency**

  Die Kombination von geringen Grenzkosten in der Produktion und Netzwerkeffekten auf der Nachfrageseite kann zur Herausbildung *von de* facto Standards führen.[41] Die Etablierung von Standards schafft für die Anbieter von Komplementärprodukten und die Nutzer eine verlässliche Technologieplattform. So ist für Softwareanbieter sichergestellt, dass Software, die nach bestimmten Kriterien programmiert ist, kompatibel zu einem definierten Standard ist. Nutzer profitieren von einem Softwarestandard, indem sie die Sicherheit haben, dass Anwendungen und Hardwarekomponenten mit einem bestimmten Standard kompatibel sind.

- **Hohe Bedeutung von Patentrechten und Lizenzen**

  Die leichte Kopierbarkeit von Software führt dazu, dass der Ausschluss von Nutzern schwierig ist. Erst durch die Vergabe von Patenten und die Lizenzierung von Software werden die rechtlichen Voraussetzungen geschaffen, Eigentumsrechte zu etablieren und durchsetzen zu können. Diese sind die Voraussetzung für eine Exkludierbarkeit von Software. Wie die Überlegungen zur Kollektivguteigenschaft von Software gezeigt haben, würde Software ohne die Festlegung von Eigentumsrechten nicht im Rahmen von Marktprozessen entstehen. Wenn eine erfolgreiche Software leicht zu kopieren ist, dann käme es zu einem kurzfristigen Preiswettbewerb und den Verlust von Marktanteilen des ursprünglich innovativen Unternehmens. Die Festlegung von Eigentumsrechten an Software ist die Voraussetzung für Unternehmen, in die Softwareentwicklung zu investieren. Investitionen sind nur dann über den Vertrieb von Software finanzierbar, wenn auch die Eigentumsrechte an der Software wirksam durchgesetzt werden können.[42]

[41] Vgl. OECD (2002), S. 105, sowie SHAPIRO UND VARIAN (1998), S. 135ff.

[42] Mit dem Kauf einer Software erhält der Käufer nicht die vollen Verfügungsrechte, sondern es erfolgt eine Einschränkung der Nutzungsrechte auf die in der Lizenz angegebenen Bereiche. Ähnlich ist es auch bei CDs oder DVDs, deren Kauf z.B. nur eine private Nutzung erlaubt. Vgl. EVANS UND REDDY (2002), S. 6 und S. 17.

2. Economics of Open Source-Software

2.1 Grundlagen von Open Source

2.1.1 Merkmale

Der Begriff Open Source-Software ist erst im Jahre 1998 entstanden.[43] Die ursprüngliche Idee einer sog. „freien" Software stammt jedoch aus den 80er Jahren. Auch wenn beide Begriffe für dieselbe Form von Software verwendet werden, betont die Bezeichnung „freie" Software eher die freiheitlichen Prinzipien im Umgang mit Software „Free software is a matter of freedom: people should be free to use software in all the ways that are socially useful."[44] Begrenzungen in der Benutzung der Software seien falsch und das letztendliche Ziel besteht darin, dass „all published software should be free software"[45].

Demgegenüber nimmt die Open Source-Initiative eine pragmatischere Position ein. Die Namensgebung Open Source-Software erfolgte vor allem, um „freie" Software breiteren, auch kommerziellen Nutzungsinteressen zu öffnen. Auch wenn der Begriff Open Source-Software umstritten ist und einen anderen Schwerpunkt setzt als „freie" Software, hat er sich im allgemeinen Sprachgebrauch etabliert und soll im Folgenden ausschließlich verwendet werden.[46]

Bestimmende Merkmale von Open Source-Software sind im Unterschied zu kommerzieller Software die an die Lizenzbedingungen gekoppelten weitgehenden Verfügungsrechte des Nutzers über die Software und die freiwillige, nicht auf Arbeitsverträgen beruhende Selbstorganisation der Entwicklungszusammenarbeit.[47] Mit Open Source-Software sind folgende Rechte für den Nutzer der Software verbunden:[48]

---

[43] Eine knappe Übersicht der Literatur zu den Economics of Open Source bietet Schiff (2002).

[44] Vgl. www.gnu.org/philosophy.

[45] STALLMANN (2001).

[46] Am Begriff Open Source-Software wird insbesondere kritisiert, dass er allein auf die technische Ausgestaltung von Software abstellt und den ursprünglichen Freiheitsgedanken vernachlässigt. Vgl. GRASSMUCK (2002), S.231-232. Die Entwicklungsgeschichte und Abgrenzung von freier Software und Open Source-Software wird in 2.1.2 dargestellt.

[47] Siehe zur Organisation der Entwicklungszusammenarbeit 2.1.3.

[48] Vgl. www.gnu.org.

- Das Recht, das Programm für jeden Zweck zu benutzen.
- Das Recht, zu verstehen, wie das Programm funktioniert und wie man es für seine Ansprüche anpassen kann.
- Das Recht, Kopien an andere Nutzer weiterzuverbreiten.
- Das Recht, das Programm zu verbessern und die Verbesserungen der Öffentlichkeit zur Verfügung zu stellen.

Aus programmiertechnischer Sicht ist die Offenheit des Quellcodes Grundlage für die Entwicklung von Open Source-Software. Nur bei Quelloffenheit und der Erlaubnis, den Quellcode auch verändern zu dürfen, ist es jedem Anwender möglich, einen eigenen Beitrag zur Weiterentwicklung der Software zu leisten. Wer den Quellcode kennt, kann neue Versionen des Programms entwickeln oder Fehler beheben sowie Verbesserungen vornehmen.[49] Im Gegensatz hierzu steht die gesetzlich geschützte kommerzielle Software, deren Quellcode vom Hersteller nicht grundsätzlich für jeden Benutzer freigegeben wird bzw. nur in Form des maschinenlesbaren Binärcodes.[50]

Tabelle 2: Softwarekategorien

			Technisches Merkmal **Offenlegung des**	
			Source Code	**Binär Code**
Ökonomisches Merkmal	**Weitergabe erfolgt**	**unentgeltlich**	Open Source-Software *Beispiele:* Linux, Apache	Freeware, Public Domain *Beispiele:* Adobe Acrobat Reader Pegasus mail
		entgeltlich	Kommerzielle Open Source-Software *Beispiele:* Open Source-Softwaredistributionen	Shareware, Kommerzielle Software *Beispiele:* Windows, MacOS

Quelle: BERLECON RESEARCH (2002c), S. 11.

[49] Vgl. MORNER (2003), S. 318.

[50] Vgl. BERLECON RESEARCH (2002c), S.11.

Unterteilt man Software nach den Kategorien Offenlegung des Quelltextes (Source Code/Binär Code) und Form der Weitergabe (entgeltlich/unentgeltlich), ergibt sich die in Tabelle 2 dargestellte Matrix, die idealtypische Softwarerubriken abbildet. Wesentliche Softwareformen sind demnach:[51]

- **Open Source-Software**

  Der Quellcode der Software ist verfügbar und die Lizenzbedingungen erlauben eine Modifizierung des Quellcodes. Die Offenheit des Quellcodes verhindert de facto eine Bepreisung (= Kommerzialisierung) der Software selbst. Programme, die den veröffentlichten Quellcode verwenden, müssen wieder unter die Lizenzbedingungen des ursprünglichen Quellcodes gestellt werden. Bekannte Beispiele für Open Source-Software sind das Betriebssystem GNU/Linux oder der Webserver Apache.

- **Kommerzielle Open Source-Software**

  Die Erstellung einer Software unter Open Source-Bedingungen schließt deren kommerzielle Verwertung durch Zusatzleistungen nicht aus. Der bloßen Verbreitung der Software gegen Bezahlung sind ökonomisch jedoch sehr enge Grenzen gesetzt, da es grundsätzlich jedem erlaubt ist, die Software unentgeltlich weiterzugeben. Deshalb bezieht sich die Kommerzialisierung immer auf Zusatzleistungen im Umfeld der Software. In diese Gruppe fallen kommerzielle Open Source-Geschäftsmodelle, die auf Open Source-Software aufbauen und über komplementäre Hardware, Dienste oder Software Erlöse erzielen.[52]

- **Freeware, Public Domain**

  Bei Freeware ist der Quellcode nicht verfügbar, sie wird in binärer Form ausgeliefert und darf nicht verändert werden. Allerdings darf Freeware unentgeltlich kopiert und weitergegeben werden.[53]

  Bei Public Domain Software verzichtet der Autor auf seine Rechte und Ansprüche. Der Nutzer hat damit uneingeschränkte Verwertungsrechte. Open Source-Software räumt demgegenüber dem Nutzer ausdrücklich Verwertungsrechte un-

[51] Vgl. BERLECON RESEARCH (2002c), S.11. Siehe auch zu einer Kategorisierung von Software LESSIG (2002), S. 52ff.

[52] Hierauf wird in Abschnitt 2.2.2 ab der Seite 46 genauer eingegangen.

[53] Mit Freeware vergleichbar sind wissenschaftliche Veröffentlichungen. Auch wissenschaftliche Veröffentlichungen werden größtenteils kostenlos verbreitet. Für Wissenschaftler ist Reputation die Bezahlung für ihre Veröffentlichung. Je größer jedoch die kommerzielle Verwertbarkeit der Forschungsergebnisse ist (etwa im Bereich der Naturwissenschaften), desto restriktiver erfolgt auch die wissenschaftliche Veröffentlichung. Vgl. EVANS UND REDDY (2002), S. 7.

ter bestimmten Bedingungen ein. Allerdings wird Public Domain Software häufig nicht im Quellcode verbreitet, so dass die Änderungsmöglichkeiten eingeschränkt sind.[54]

- **Kommerzielle Software**
  Die Verwertung von kommerzieller Software erfolgt im Normalfall nur ohne Offenlegung des Quellcodes. Kommerzielle Software wird in lizenzierter Form vertrieben. Mit der Lizenz werden für den Nutzer bestimmte Nutzungsbedingungen festgeschrieben. Diese schließen in der Regel eine Mehrfachverwendung oder Weitergabe der Software aus. Da der Quellcode nicht zugänglich ist, bestehen auch nicht die technischen Voraussetzungen, die Software zu verändern.

  Diese vier Softwarekategorien bilden idealtypische Gruppen ab. In der Realität lassen sich diese Gruppen nicht immer so trennscharf abgrenzen. Vielfältige Finanzierungsformen und auch Möglichkeiten der Einsichtnahme in den Quellcode sind vorhanden. So gibt es Software, die kostenlos genutzt werden darf, die jedoch zu einer freiwilligen Bezahlung in Spendenform aufruft. Shareware kann ebenfalls für einen bestimmten Zeitraum kostenlos genutzt werden, muss dann allerdings bezahlt werden. Auch bei kommerzieller Software, die nur im binären Code vertrieben wird, kann eine Offenlegung des Quellcodes erfolgen. So legt beispielsweise Microsoft im Rahmen seiner Shared Source-Initiative für Kunden, Partner und Regierungen einige Teile des Quellcodes offen.[55]

2.1.2 Entstehung und Lizenzmodelle

Zu Beginn der Verbreitung von Computern existierte kein eigenständiger Markt für Software. Computer waren Großrechner, die nur in großen Unternehmen oder speziellen Rechenzentren an Universitäten eingesetzt wurden. Bedient werden konnten diese Computer nur von Spezialisten, die auch Eingriffe in die Software vornehmen konnten oder eigene Software entwickelten und gegenseitig austauschten. Software wurde zunächst ausschließlich für den eigenen Gebrauch entwickelt, speziell für verschiedene Typen von Hardware und damit äußerst individuell. Einen Massenmarkt für standardisierte Software konnte es nicht geben, denn die entsprechende Hardware war in Größe und Preis nicht für den Massenmarkt konzipiert.

---

[54] Vgl. SPINDLER (2003), S. 18 und HANG UND HOHENSOHN (2003), S. 7.

[55] Vgl. O. V. (2003).
Für einen Überblick siehe www.microsoft.com/resources/sharedsource/Initiative/Initiative.mspx.

Die Aktivitäten der Entwickler bezogen sich auf die Programmierung von Software für die von ihnen benutzten Computer. Sie hatten starke Anreize, ihre Arbeit mit anderen Entwicklern zu teilen, die dieselbe Hardware nutzten. Weil Software nur als notwendiges Komplement zur Hardware vertrieben wurde und die Entwickler auf den Austausch untereinander angewiesen waren, war der Quellcode der Software offen und die Software frei verfügbar. Auch wenn die Software auf verschiedenen Rechnern installiert wurde, so hatte sie doch deutliche Merkmale einer Individualsoftware. Zwar waren die Programme nicht kostenpflichtig, die Entwickler wurden jedoch für das Programmieren bezahlt.[56]

In dieser Entwicklungsumgebung wurde im Jahre 1969 das Betriebssystem Unix von zwei Mitarbeitern des Telekommunikationskonzerns AT&T programmiert. Der Unix-Quellcode lag offen und konnte von anderen Entwicklern weiterentwickelt werden. Ohne von einem Unternehmen gefördert oder beworben zu werden, verbreitete sich Unix schnell. Nach der Aufspaltung des Telekommunikationsmonopols von AT&T, begann AT&T Unix eigenständig zu vermarkten. Mittlerweile waren - bedingt durch den freien Zugriff auf den Quellcode und seine Veränderbarkeit - verschiedene inkompatible Versionen von Unix entstanden.

Eine der ersten Lizenzen für freie Software war die **Berkeley Software Distribution-Lizenz** (BSD-Lizenz). An der Universität von Kalifornien in Berkeley wurden auf UNIX-Basis Programme entwickelt und als Berkeley Software Distribution vertrieben. Um unabhängig von der AT&T-Lizenzierung die an der Universität Berkeley entstandenen Programmelemente vertreiben zu können, wurde die BSD-Lizenz geschaffen. Sie erlaubt die Verwendung, Veränderung und den kostenlosen Vertrieb des Quell- oder Binärcodes. Einzige Bedingung für die Weiterverwendung ist, dass in allen veränderten und weiterverbreiteten Programmen der Urheberrechtsverweis auf die Universität von Kalifornien enthalten sein muss.[57]

Allerdings wird mit der BSD-Lizenz zwar festgelegt, dass auf ihr basierende Software wieder unter die freien Lizenzbedingungen fallen, also ebenfalls verändert werden dürfen. Es wird aber nicht explizit festgelegt, dass die veränderte Software auch im Quellcode erhältlich sein muss. Damit sind die praktischen Möglichkeiten

[56] Vgl. zur Entwicklungsgeschichte auch GRASSMUCK (2002), S. 202ff.

[57] Vgl. HANG UND HOHENSOHN (2003), S. 26-27, EVANS UND REDDY (2002), S. 8-9. Die Bedingung des Urheberrechtsvermerks wurde 1999 aufgehoben.

einer Veränderung eingeschränkt und die kostenpflichtige Abgabe aufbauender Programme lässt sich leichter durchsetzen.[58]

Die Kommerzialisierung und Durchsetzung von Eigentumsrechten an Software, die der ursprünglichen freien Programmierung, Veränderung und Weitergabe von Software entgegen stand, führte 1984 zum GNU-Projekt, dem eigentlichen Start der Entwicklung von Software, deren Erstellung und Vertrieb bewusst als frei eingestuft wurden. Zielsetzung war es, ein unix-ähnliches Betriebssystem zu schreiben, das quelloffen ist und in freiwilliger Zusammenarbeit weiterentwickelt werden kann. Zur Distribution der GNU-Software wurde 1985 die Free Software Foundation FSF gegründet. Um die entstehende freie Software vor der Kommerzialisierung zu schützen, wurde 1989 von der Free Software Foundation die **General Public Licence** (GPL) veröffentlicht. Diese beinhaltet die in Abschnitt 2.1.1 aufgezeigten vier Rechte.

Der Copyright Schutz der GPL ist notwendig, um eine kommerzielle Verwertung der Software zu verhindern. Ohne die Lizenzbedingung, dass Software, die den Quellcode verwendet, wieder unter die Bedingungen der GPL gestellt werden muss, könnten kommerzielle Softwareanbieter den freien Quelltext kostenlos in ihre Software einbauen. Damit schützt die GPL das geistige Eigentum an der freien Software und verlangt als Gegenleistung für dessen Nutzung ebenfalls eine freie Zugänglichkeit des Quellcodes.[59] Die Verwendung der Software ist damit nicht grundsätzlich frei, sondern nur insofern, wie sie innerhalb der von den Lizenzbedingungen festgelegten Grenzen bleibt.

Wichtiger Bestandteil der GPL ist der so genannte „Virus-Effekt". Er ergibt sich aus folgender Anforderung: „*You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the program or any part there of, to be licensed as a whole at no charge to all third parties under the terms of this license*".[60] Im Gegensatz zu „Copyright" benutzt die Free Software Foundation den Begriff „Copyleft" als Bezeichnung für diese Methode, Softwareprogramme frei zugänglich zu machen und eine kommerzielle Weitergabe zu verhindern. Die Copyleft-Anforderung führt dazu, dass Software, die in Teilen auf unter GPL-Bedingungen veröffentlichte Software zurückgreift, wieder unter die Bedingungen der GPL ge-

[58] Vgl. GRASSMUCK (2002) S. 216-217 und S. 279-280 und SCHMIDT UND SCHNITZER (2003), S. 5.

[59] Vgl. LERNER UND TIROLE (2000), S. 5-6 und SCHMIDT UND SCHNITZER (2003), S. 5.

[60] Vgl. www.fsf.org/copyleft/copyleft.html.

stellt werden muss. Damit ist eine kommerzielle Verwertung von Software oder von Bestandteilen des Quellcodes in anderen Programmen, also das Erzielen von Erlösen durch den Verkauf der Programme, ausgeschlossen.[61]

Eine Abwandlung der GPL ist die LGPL (**Library General Public License** seit 1999 Lesser General Public License).[62] Sie erlaubt es, dass Programme, die mit einer der LGPL unterliegenden Bibliothek verbunden werden, nicht als abgeleitete Werke im Sinne der GPL zu sehen sind. Innerhalb eines Programms lässt die Verwendung von Bibliotheken die Grenzen zwischen Gebrauch und Veränderung verschwimmen, so dass nicht eindeutig festlegbar war, wann der Zugriff auf eine GPL-Bibliothek dazu führt, dass das abgeleitete Programm wieder unter die GPL fallen muss. Die LGPL erlaubt eine solche Verwendung von Bibliotheken, ohne dass das abgeleitete Programm wieder unter die GPL fallen muss. Software, die unter der LGPL veröffentlicht wurde, bietet für bestimmte Fälle damit bessere Möglichkeiten einer Verbindung zwischen freier und kommerzieller Software.

Einhergehend mit der weiteren Verbreitung und der Entwicklung von neuen Open Source-Programmen veränderte sich auch die Lizenzierungsmethode. Während in den achtziger Jahren die in ihrer Auslegung sehr restriktive GPL die vorherrschende Software-Lizenz war, entwickelte sich in den neunziger Jahren eine Tendenz zu flexibleren, weniger restriktiven Lizenzvereinbarungen. Für die Verwendung von liberaleren Lizenzen sprechen sich vor allem die Entwickler aus, die kommerzielle Aktivitäten und die Verwendung von geschütztem Code in bestimmten Segmenten gutheißen, um so eine breitere Versorgung mit Open Source-Software zu gewährleisten.[63]

Ende der 90er Jahre kam es zu dem bereits angesprochenen Versuch, den Begriff Open Source-Software als Alternative zu freier Software zu etablieren. Im Jahre 1997 wurde die **Open Source Initiative** OSI gegründet. Die von der OSI entwickelte Open Source-Definition ist keine eigenständige Lizenz, sondern ein Gütesiegel zur Bewertung von Lizenzen. Erfüllt eine Lizenz die in der Open Source-

[61] Allerdings wird auf die Möglichkeit des „dual-licensing" verwiesen. Danach kann ein Copyright-Inhaber eine Software generell unter den Bedingungen der GPL veröffentlichen und für bestimmte Nutzer diese Software unter andere als die GPL-Bedingungen stellen. Diese Möglichkeit wird von der FSF jedoch nur sehr restriktiv gehandhabt. Siehe www.fsf.org/copyleft/gpl-faq.html.

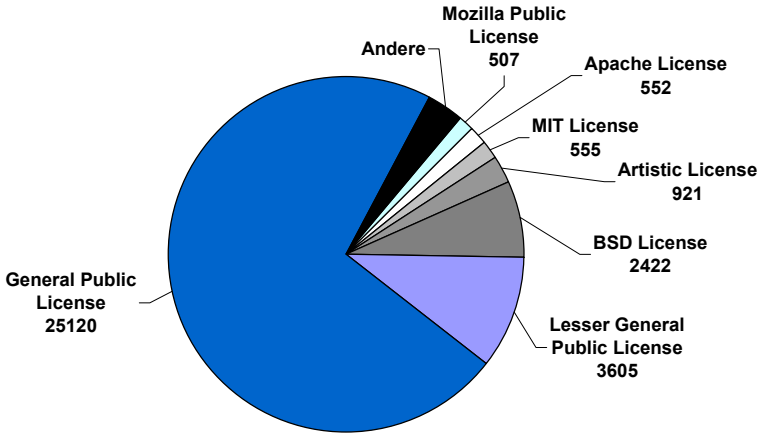
[62] Vgl. GRASSMUCK (2002), S. 289-293 und HANG UND HOHENSOHN (2003), S. 26.

[63] Vgl. LERNER UND TIROLE (2000), S. 7. Zu den Problemen, die sich aus liberaleren Lizenzen ergeben können siehe LERNER UND TIROLE (2000), S. 30.

Definition angegebenen Kriterien, so darf sie den geschützten Titel „Open Source" tragen.[64]

Die Open Source-Initiative verfolgte die Strategie, Open Source zu popularisieren und dem Anti-Business-Image der freien Software und der Free Software Foundation entgegenzutreten.[65] Diese Überlegungen führten zum Begriff Open Source-Software als Alternative für freie Software. Die Open Source Initiative veröffentlicht auf ihrer Webseite eine Liste der wichtigsten Lizenzen und gibt in einem Kriterienkatalog an, wann eine Lizenz als Open Source zu beurteilen ist.[66]

Abbildung 4: Verteilung der Lizenzen



Quelle: BERLIOS (2003).

Neben diesen hier beschriebenen Lizenzen gibt es eine Vielzahl an Lizenzen, die für bestimmte Projekte geschaffen wurden.[67] Die Abbildung 4 zeigt die Verteilung

[64] Vgl. MENDYS-KAMPHORST (2002), S. 10 sowie HANG UND HOHENSOHN (2003), S. 14.

[65] Ein populärer Vergleich um das Image von freier Software zu verdeutlichen, ist die Gegenüberstellung von „freier Rede" und „Freibier". Viele Unternehmen verbanden den Begriff „freie Software" eher mit „Freibier" als mit „freier Rede" und zögerten daher, ein Betriebssystem einzuführen, dass an jeden freizügig verschenkt wird.

[66] www.opensource.org/licenses/ und www.opensource.org/docs/definition.php Vgl. auch WEBER (2000), S. 10f.

[67] Vgl. Für eine Übersicht und Einordnung der verschieden Lizenzen siehe www.ifross.de/ifross_html/lizenzcenter.html.

der Lizenzen im Januar 2003 innerhalb der Sourceforge.org Datenbank.[68] 89 Prozent der dort aufgeführten Projekte sind unter der GPL, LGPL oder BSD lizenziert.

Die wichtigsten bzw. bekanntesten Lizenzen können der nachfolgenden Tabelle entnommen werden. Dabei wird Freeware nur zum Vergleich mit Open Source-Software angegeben.

Tabelle 3: Wichtige Lizenzen und ihre wesentlichen Bedingungen

	Kostenlose Nutzung	Quellcode veränderbar	Quellcode muss bei Derivaten offenliegen	Verknüpfung mit geschützten Elementen nicht möglich
Freeware	X			
BSD	X	X		
LGPL	X	X	X	
GPL	X	X	X	X

Quelle: In Anlehnung an SPINDLER (2003), S. 19 und BERLECON RESEARCH (2002c), S. 16 mit eigenen Änderungen.

2.1.3 Arbeitsorganisation und Entwicklungszusammenarbeit

Die Entwicklungszusammenarbeit bei der Erstellung von Open Source-Software kann in unterschiedlichen Organisationsformen erfolgen. In der Regel schließen sich Softwareentwickler freiwillig einem Open Source-Projekt an und sind je nach geleistetem Beitrag unterschiedlich intensiv in das Softwareprojekt und die Entscheidungsstrukturen eingebunden. Solche Softwareprojekte finden sich spontan und selbstorganisiert zusammen. Die Mitarbeit an einem Softwareprojekt geschieht freiwillig und ohne finanzielle Gegenleistung.[69]

Im Gegensatz zur traditionellen Software-Entwicklung steht bei Open Source-Projekten nicht die Identifikation der Kundenbedürfnisse am Anfang, sondern in den meisten Fällen die Idee oder das konkrete Problem eines Entwicklers.[70] Der Start-

[68] Sourceforge.org stellt eine Datenbank von Open Source Projekten zur Verfügung. Eine Untersuchung der verschiedenen Lizenzmodelle bei sourceforge.org liefern LERNER UND TIROLE (2002).

[69] Schon an dieser Stelle zur Abgrenzung: Die kostenlose Mitarbeit bezieht sich auf die Erstellung von allgemein verwendbarer Standardsoftware. Auch auf Open Source Basis ist die Programmierung von Individualsoftware möglich. Dabei erfolgt eine Bezahlung der Entwickler und nicht der Software. Hierauf wird später noch genauer eingegangen.

[70] Vgl. SMITH (2002), S. 72.

zeitpunkt eines Open Source-Projektes ist die Veröffentlichung des Projektes und des Quellcodes. Erscheint das Projekt interessant, finden sich weitere Entwickler, die sich dem Projekt anschließen. Wächst ein Projekt, können Strukturen entstehen, bei denen ein zentraler Maintainer oder ein Core-Team grundlegende Entscheidungen über die weitere Entwicklung fällen. Aus größeren Projekten können Module herausgelöst werden, für die es wiederum verantwortliche Maintainer gibt.[71]

Unterhalb der Maintainer oder des Core-Teams sind vielfältige Formen der Mitarbeit an einem Softwareprojekt möglich. Diese kann vom einfachen Testen des Programms über Hinweise für neue Funktionen bis zum Bug-Fixing und der eigentlichen Mitarbeit am Quellcode reichen. Die Kommunikation erfolgt über E-Mail, Mailinglisten und Newsgroups.[72]

Ein häufig zitiertes Beispiel zur Verdeutlichung der Entwicklungszusammenarbeit ist der Vergleich zwischen einem Basar und Kathedralenbau. Die Entstehung von Open Source-Software wird dabei mit einem Basar verglichen, während kommerzielle Software als Kathedralenbau angesehen wird. Open Source-Software sei ein „great babbling bazaar of differing agendas and approaches out of which a coherent and stable system could seemingly emerge only by a succession of miracles".[73] Im Gegensatz zu diesem „bazaar" wird die Entwicklungsform kommerzieller Software als „cathedral" und damit als Inbegriff einer hierarchisch-strukturierten, durch strenge Autorität geprägte Form der Arbeitsteilung aufgefasst.

Mit dem freiwilligen Zusammenschluss zur Entwicklungsarbeit ist jedoch auch die Gefahr verbunden, dass Projekte „austrocknen" können. Wenden sich die Entwickler neuen, aus ihrer Sicht interessanteren und evtl. visibleren Projekten zu, kann es dazu kommen, dass Projekte aufgrund fehlender Unterstützung nicht zu Ende geführt werden. So erreichen von den aktuell bei sourceforge.net registrierten Programmen nur 16% den Development Status „Production/Stable" und nur 2% den Status „Mature".[74]

---

[71] Vgl. MENDYS-KAMPHORST (2002), S. 13.

[72] Vgl. MORNER (2003), S. 320.

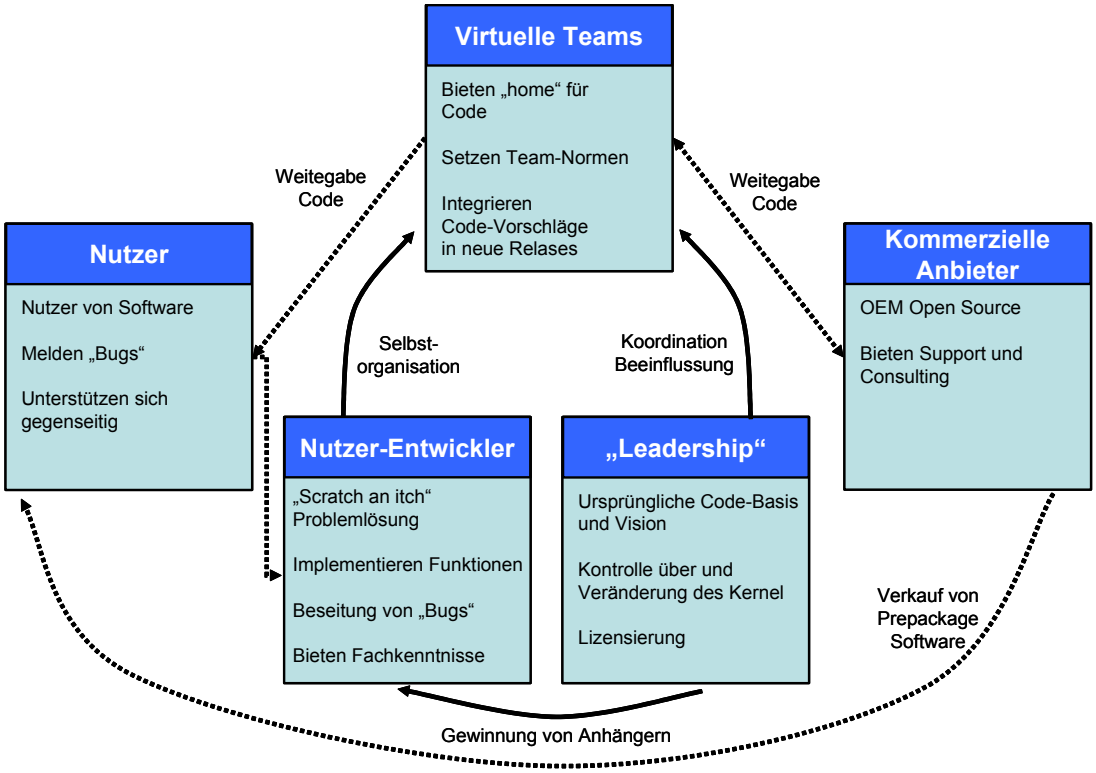
[73] Vgl. RAYMOND (1998a).

[74] Vgl. www.sourceforge.net unterscheidet den Development Status nach 1. Planning , 2. Pre-Alpha, 3. Alpha, 4. Beta, 5. Production/Stable, 6. Mature und 7. Inactive. Am 27.10.2003 finden sich folgende Angaben: 1. Planing 13.248 (26%), 2. Pre-Alpha 9.388 (18%), 3. Alpha 8.833 (17%), 4. Beta 10.739 (21%), 5. Production/Stable 8.503 (16%), 6. Mature 823 (2%), 7. Inactive 327 (1%).

Die Entwicklung von Open Source-Software muss allerdings nicht zwangsläufig in dieser selbstorganisierten Form erfolgen. Open Source-Software kann auch innerhalb von Unternehmen von fest angestellten Entwicklern geschrieben werden. So beschäftigen beispielsweise Distributoren häufig eigene Entwickler, um Anpassungen an bestimmte Entwicklungsstufen einer Software vorzunehmen. Wesentlich ist dabei, dass auch der Beitrag, den die fest angestellten Entwickler bei Unternehmen leisten, allen anderen Nutzern dieser Software zugänglich gemacht werden muss.

Die vernetzte Zusammenarbeit an verschiedenen Teilproblemen wird durch einen modularen Aufbau der Projekte ermöglicht. Dies führt dazu, dass sich die Funktionalität von Programmen häufig auf einzelne Teilaufgaben bezieht. Erst im Zusammenspiel verschiedener Module und Programme ergibt sich dann der gewünschte Funktionsumfang.

Abbildung 5: Schematische Darstellung der Funktionsweise eines Open Source-Projektes



Quelle: BOSTON CONSULTING GROUP (2002), S. 8 mit eigenen Änderungen.

Die Abbildung 5 zeigt in einer schematischen Darstellung die Zusammenarbeit innerhalb eines Open Source-Projektes. Zu beachten ist dabei schon an dieser Stelle, dass die Nutzer von Software nur dann eine Möglichkeit haben, ihre Vorstellungen in die Software einfließen zu lassen, wenn sie selbst zu Entwicklern werden. Inner-

halb des Open Source-Modells gibt es keinen Rückkanal, über den die Nutzer von Software, die nicht selbst Entwickler sind, ihre Bedürfnisse zum Ausdruck bringen können. Auf dem kommerziellen Softwaremarkt können sie dies durch ihre Kaufentscheidung erreichen.

Open Source-Projekte können jedoch auch aus Programmen entstehen, deren Quellcode zunächst nicht zugänglich war. So öffnete Netscape 1998 den Quellcode seines Internetbrowsers „Netscape Communicator" in der Hoffnung, durch den Einsatz freier Programmierer dessen Entwicklung voranzutreiben und verlorene Marktanteile zurückzugewinnen. Eine ähnliche Entwicklung hat das Star Office Softwarepaket genommen, das zunächst ebenfalls als kommerzielle Software vertrieben und dessen Quellcode später freigegeben wurde.

Im Falle von Netscape wurde nicht der gesamte Quellcode freigegeben, sondern nur einzelne Teile. Zudem bestand Netscape zunächst auf Lizenzbedingungen, die eine spätere Aneignung des freigegebenen Codes wieder ermöglicht hätte. Dies begrenzte die Bereitschaft, am Mozilla-Projekt mitzuarbeiten.[75] Infolgedessen arbeiteten größtenteils bezahlte Mitarbeiter am Mozilla-Projekt, es sollen nicht mehr als zwei Dutzend externe Entwickler am Projekt beteiligt gewesen sein. Auch die Leitung des Projektes durch eine kommerzielle Firma kann die Beteiligungsbereitschaft in diesem Fall begrenzt haben.[76] Erst durch die Anpassung der neuen Lizenzstruktur war die Veröffentlichung einer stabilen Version möglich, die insgesamt vier Jahre in Anspruch nahm.[77]

Open Source-Projekte, die aus einer zuvor kommerziellen Software entstehen, haben zudem das Problem, dass sich für einen gegebenen und fertig gestellten Quellcode eine Entwicklergemeinde finden muss. Dabei ist es deutlich schwieriger, sich in den Quellcode eines fertigen Produktes hineinzuarbeiten und diesen weiterzubearbeiten, als selbständig ein eigenes neues Projekt zu entwickeln, bei dem die gesamte Entwicklungsgeschichte offen gelegt ist.[78] Zudem ist bei diesen Open Source-Projekten zu bedenken, dass der Quellcode von ehemals kommerzieller Software nur als „Restverwertung" geöffnet wird, wenn sich für die kommerzielle Version

[75] Siehe hierzu: www.gnu.org/philosophy/netscape-npl.html.

[76] Vgl. LERNER UND TIROLE (2000), S. 28.

[77] Vgl. OSTERLOH, KUSTER, UND ROTA (2002), S. 15.

[78] Vgl. GRASSMUCK (2002), S. 257.

nicht genügend Nutzer gefunden haben und die Software somit betriebswirtschaftlich nicht erfolgreich war.

2.2 Geschäftsmodelle und Marktübersicht

2.2.1 Die Software-Wertschöpfungskette – ein Vergleich zwischen Open Source-Software und kommerzieller Software

Die Unterschiede in der Entwicklungszusammenarbeit zwischen kommerzieller und Open Source-Software führen zu Unterschieden in der Abdeckung der Softwarewertschöpfungskette zwischen beiden Modellen. In der Wertschöpfungskette schließen sich an die Softwareentwicklung die Software-Dienste an. Bestandteile der Software-Dienste sind Consulting, Implementation, Support, Training und Application Management.[79]

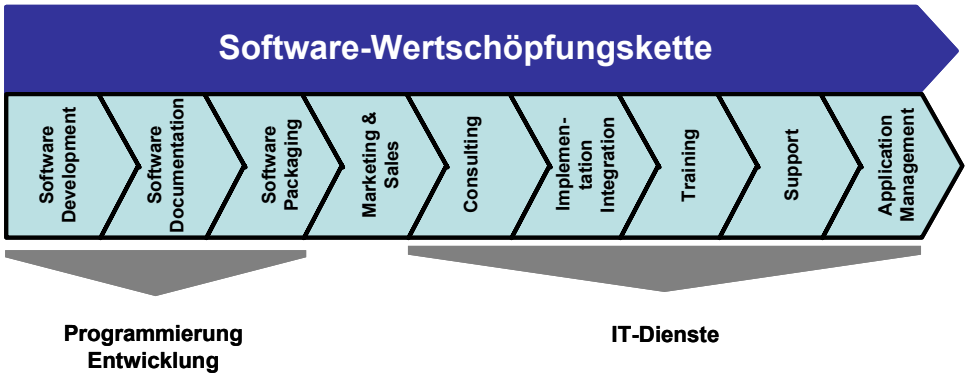
Auch wenn die einzelnen Elemente der Software-Wertschöpfungskette bei Standard- und Individualsoftware identisch sind, unterscheidet sich die Reihenfolge, in der sie angewendet werden. So steht im Gegensatz zur Standardsoftware bei Individualsoftware das Consulting vor der eigentlichen Programmierarbeit. Ebenso gibt es Unterschiede in der Wertschöpfungskette zwischen kommerzieller und Open Source-Software. Zwar finden sich bei beiden alle Elemente der Wertschöpfungskette, allerdings gibt es eine unterschiedliche Gewichtung der einzelnen Schritte bei rein kommerziellen und kommerziellen Open Source-Geschäftsmodellen. Zudem lassen sich mit einigen Elementen der Wertschöpfungskette im Open Source-Modell keine Erlöse erzielen. Hierauf wird im weiteren Verlauf der Studie noch genauer eingegangen.

Am Beginn der Wertschöpfungskette steht die Softwareentwicklung. Bestandteil der Softwareentwicklung ist bei kommerzieller Software das Festlegen der Anforderungen an die Software auf Basis der erwarteten Kundenwünsche. Des Weiteren wird in diesem Schritt das Design der Software bestimmt und der Quelltext geschrieben. An die Produktion schließen sich die Dokumentation und das Packaging der Software an. Beim Packaging werden einzelne Softwareprodukte zu einem vertriebsfertigen Gesamtpaket zusammengestellt. Die Softwareentwicklung, die Dokumentation und das Packaging bilden zusammen die Softwareproduktion. Während die Dokumentation und das Packaging nur ein Bestandteil des Produktionspro-

[79] Vgl. hier und zur Wertschöpfungskette insgesamt BERLECON RESEARCH (2002c), S. 22-29.

zesses bei kommerzieller Software sind, konzentrieren sich einzelne Open Source-Geschäftsmodelle (Distributoren) nur auf diese Elemente.

Abbildung 6: Die Software-Wertschöpfungskette



Quelle: BERLECON RESEARCH (2002c), S. 23.

Aufgabe des Marketings ist es, den Instrumenteneinsatz innerhalb des Marketing-Mix zu optimieren und so die Bekanntheit, die Akzeptanz und damit den Absatz des Produktes zu fördern. Unterschiede zwischen kommerzieller Software und Open Source-Software ergeben sich in allen Elementen des Marketing-Mix. Insgesamt sind die Möglichkeiten, über das Marketing auf das Produkt und den Absatz Einfluss zu nehmen innerhalb des Open Source-Modells geringer als im Fall der kommmerziellen Software.

Bei der Produktgestaltung, als erstem Element des Marketing-Mix, gibt es aufgrund des spezifischen Open Source-Entwicklungsprozesses keine Einflussmöglichkeiten. Bei kommerzieller Software können hier z. B. der Markteintrittszeitpunkt, die Qualität oder bestimmte Kundenanforderungen festgelegt werden. Wesentliche Unterschiede zwischen Open Source-Software und kommerzieller Software bestehen insbesondere beim Marketing Instrument Preis. Open Source-Software kann nur dann bepreist werden, wenn z. B. durch das Packaging eine über das Programmieren hinausgehende zusätzliche Funktionalität hinzugefügt werden kann. Da die Software selbständig zusammengestellt werden könnte bzw. eine verkaufte Softwarezusammenstellung auch auf verschiedenen Rechnern verwendet werden kann, ist der Spielraum für die Preisgestaltung bei Open Source-Softwaredistributionen wesentlich geringer als bei kommerzieller Software.

Weitere Elemente des Marketing-Mix sind die Distribution und das Advertising. Die Zusammenstellung und Distribution eines bestimmten Softwarepaketes ist der wesentliche Bestandteil des Marketings bei Open Source-Software.

An das Marketing schließt sich das Consulting als Bestandteil des Software-Services an. Im Rahmen des Consultings wird die spezifische Situation eines Unternehmens analysiert, es werden die Anforderungen an die einzusetzende Software festgelegt und die Software ausgewählt. Unterschiede zwischen Open Source-Software und kommerzieller Software bestehen hierbei nicht.

Im Rahmen der Implementierung wird die Software installiert und an die gegebenen Anforderungen angepasst. Dabei ist z. B. die Zusammenarbeit innerhalb eines Netzwerkes oder mit verschiedenen Anwendungen zu beachten. Open Source-Software bietet hier im Vergleich zu kommerziellen Standardpaketen eine nahezu unbegrenzte Anpassungsfähigkeit.

Im Rahmen des Trainings werden Anwender mit der installierten Software vertraut gemacht. Bezüglich des Trainingsangebotes bestehen keine grundsätzlichen Unterschiede zwischen kommerzieller und Open Source-Software.

Support ist notwendig, wenn der Nutzer mit der Verwendung der installierten Software Probleme hat. Anbieter von kommerzieller Software bieten ein weites Spektrum verschiedener Formen des Supports an. Dies kann von FAQs über Newsgroups bis zu eigenen Hotlines reichen. Dabei sind Vertriebsmodelle möglich, die unterschiedliche Formen des Supports beinhalten. Der Support bei Open Source-Software wird primär über die „Community" geleistet, die die Software erstellt hat. Daneben bieten aber auch Distributoren oder unabhängige Beratungsfirmen Support für Open Source-Software an.

Das letzte Element in der Wertschöpfungskette ist das Application Management. Es umfasst alle Bestandteile, die zum ordnungsgemäßen Betrieb der Software notwendig sind. Hierzu gehören auch mögliche Updates oder Backups. Das Application Management kann innerhalb des Unternehmens von Systemadministratoren durchgeführt oder aber extern vergeben werden.

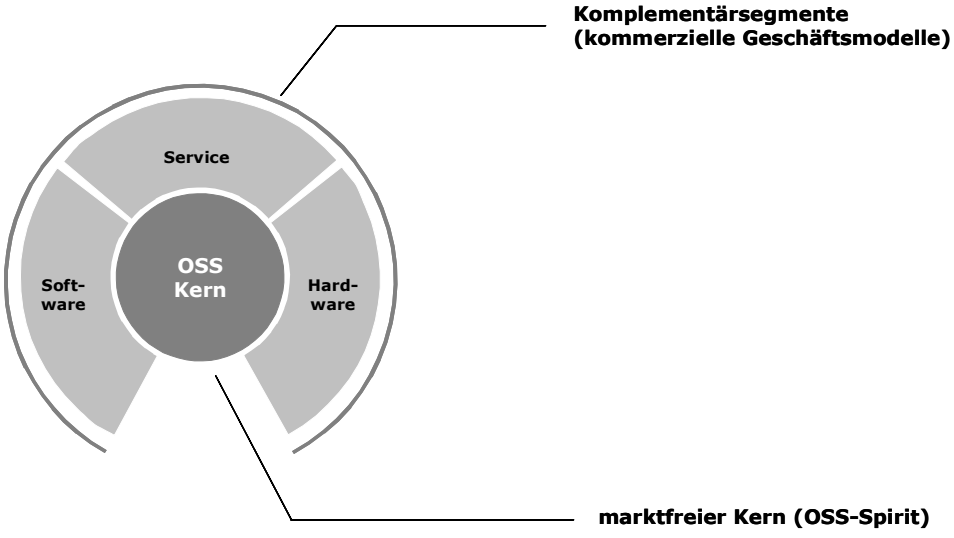
2.2.2 Open Source-Software als Basis für Geschäftsmodelle

Eine wichtige Antriebsfeder für die Entstehung von freier Software ist es, eine Aneignung der Eigentumsrechte an der Software und damit ihre kommerzielle Verwertung zu verhindern. Darauf zielen auch die damit verbundenen Lizenzmodelle ab.

Da Open Source-Software legal kopiert und z. B. aus dem Internet heruntergeladen werden darf, scheidet eine kommerzielle Verwertung der Software über kostenpflichtige Lizenzmodelle aus. Einen Markt, auf dem Anbieter und Nutzer der so entwickelten Software zusammenkommen, gibt es daher nicht.

Dennoch sind rund um Open Source-Software kommerzielle Geschäftsmodelle entstanden. Ihnen ist gemeinsam, dass sie auf Open Source-Software aufbauen und diese als Basis für eigene Zusatzdienste oder -produkte nutzen. Alle Geschäftsmodelle versuchen, über ihr Open Source-Engagement die Nachfrage nach einem eigenen komplementären Produkt zu erhöhen. Aus betriebswirtschaftlicher Sicht lohnen sich Investitionen für Unternehmen in Open Source-Projekte nur dann, wenn sich durch diese Investitionen Gewinne mit anderen Diensten oder Produkten erwirtschaften lassen. Weil sich mit Verbesserungen an der Open Source-Software keine direkten Erlöse erzielen lassen, müssen Firmen indirekt über andere Produkte und Dienste von der Open Source-Entwicklung profitieren. Open Source-Businessmodelle können daher als indirekte Businessmodelle bezeichnet werden, denn Umsätze werden nicht mit dem eigentlichen Produkt erwirtschaftet, sondern mit zusätzlichen, darauf aufbauenden Produkten und Diensten.[80]

Abbildung 7: Open Source-Kern und indirekte Open Source-Geschäftsmodelle



.

[80] Vgl. RAYMOND (1998b), LERNER UND TIROLE (2000), S. 26 und SCHMIDT UND SCHNITZER (2003), S. 12.

Auch wenn eine Erlöserzielung nur indirekt möglich ist, bleibt festzuhalten, dass im Gegensatz zur Entwicklergemeinde bei Unternehmen rein ökonomische Motive vorliegen, an Open Source-Projekten zu partizipieren. Die Abbildung 7 zeigt verschiedene Möglichkeiten, auf Open Source-Basis kommerzielle Geschäftsmodelle aufzusetzen. Dabei wird unterschieden nach Geschäftsmodellen, die zusätzliche Dienste, zusätzliche Software oder zusätzliche Hardware anbieten. Allen diesen Modellen ist gemeinsam, dass sie sich des Software- und Entwicklerpools bedienen und der Open Source-Software eigene Produkte und Dienste zufügen. Limitiert sind dabei jedoch die Anreize der Unternehmen selbst, in die Entwicklung von Software zu investieren. Die Investitionen in Open Source-Software können nicht mit Erlösen aus dem Verkauf dieser Software finanziert werden. Daher müssen alle Ausgaben für die Entwicklung über die zusätzlichen Angebote erwirtschaftet werden. Auf die einzelnen Geschäftsmodelle und die limitierenden Faktoren wird im Folgenden genauer eingegangen.[81]

2.2.2.1 Verkauf zusätzlicher Dienste

Distributoren von Open Source-Software bieten als zusätzlichen Dienst die Zusammenstellung, das Testen und die Abstimmung dieser Software an. Bekannte Linux-Distributoren sind z. B. Red Hat, SuSE oder MandrakeSoft. Der Endnutzer kann für verschiedene Zwecke (z. B. Server Anwendungen, Desktop-Anwendungen, Software für Administratoren oder Entwickler) unterschiedliche Zusammenstellungen von Linux-Software kaufen. Die unterschiedlichen Versionen verschiedener Distributoren müssen dabei nicht untereinander kompatibel sein.

Distributoren bedienen sich aus dem gegebenen Open Source-Softwarepool und übernehmen mit der Zusammenstellung und Abstimmung der Software untereinander anschließend eine Aufgabe, die bei kommerzieller Software bereits von dem entwickelnden Unternehmen geleistet wird. Eine Linux-Distribution enthält den Linux-Kernel sowie eine Vielzahl von zusätzlichen Komponenten, die zusammen das Linux-Betriebssystem bilden. Um eigene Versionen entwickeln zu können, werden zunächst die aktuellsten Versionen einzelner Komponenten zusammengetragen (sog. packaging) und in einem zweiten Schritt getestet, abgestimmt und optimiert. Zum Abschluss wird die Distribution dokumentiert und soweit eingerichtet, dass eine einfache Installation erfolgen kann. Dass Distributoren notwendig sind, um

[81] Vgl. Eine umfangreiche Übersicht der Open Source-Software-Aktivitäten verschiedener Unternehmen bietet BERLECON RESEARCH (2002b).

Abstimmungen und Anpassungen an der Software vorzunehmen, bevor sie von breiten Nutzerkreisen verwendet werden kann, ist auch ein Zeichen für die mangelnde Eignung des Open Source-Entwicklungsprozesses, für den Endkunden unmittelbar verwendbare Produkte hervorzubringen.[82]

Für den Käufer einer Linux-Distribution entfällt die Notwendigkeit, die Software selbst zu suchen, herunterzuladen und so anzupassen, dass die einzelnen Komponenten miteinander fehlerfrei arbeiten. Mit einer Distribution erwirbt der Käufer auch die Möglichkeit, in bestimmten Abfolgen Updates oder Bugfixes zu erhalten. Grundsätzlich könnte jeder Nutzer auch selbständig diese Komponenten zusammenstellen, installieren und auftretende Fehler selbst beheben.

Um die notwendigen Anpassungen an der Software vorzunehmen, beschäftigen die Distributoren eigene Entwickler. Jeder Entwicklungsbeitrag eines Distributors an seiner eigenen Linux-Distribution muss jedoch wiederum frei zugänglich gemacht werden. Linux-Distributoren sparen zwar den größten Teil der Entwicklungsarbeiten an der vertriebenen Software und profitieren von einer weiten Verbreitung der Software. Der von ihnen geleistete Entwicklungsbeitrag kommt jedoch nicht ihnen allein zugute, sondern auch allen anderen in diesem Markt tätigen Unternehmen. Damit unterstützen sie mit ihrer Entwicklungsarbeit auch direkt mögliche Wettbewerber, die als „Trittbrettfahrer" ebenfalls von den Entwicklungsinvestitionen profitieren.[83]

Bei der Bepreisung ihrer Produkte ist der Spielraum der Distributoren sehr eingeschränkt. Nicht nur können die einzelnen Softwarekomponenten kostenlos selbst zusammengestellt werden, auch die kostenpflichtig vertriebenen Distributionen dürfen von Käufern wiederum kostenlos weitergegeben werden. Ebenso können andere Unternehmen eine bestimmte Distribution übernehmen und als eigene Distribution selbst vertreiben. Wenn der Quellcode offen gelegt ist und jeder Empfänger ihn einsehen kann, kann die Software sehr einfach weiter vertrieben werden. Dies übt solang einen Preisdruck aus, bis der Vertriebspreis auf Höhe der durchschnittlichen Vertriebskosten liegt.[84] Die durchschnittlichen Vertriebskosten sind in diesem Fall für den Software-Distributor die Preisuntergrenze, denn sofern er keine Entwicklungskosten hat, fallen bei ihm lediglich die Vertriebskosten an. Der kostenlose

[82] Vgl. LERNER UND TIROLE (2000), S. 26.

[83] Vgl. LERNER UND TIROLE (2000), S. 27.

[84] Vgl. SCHMIDT UND SCHNITZER (2003), S. 3.

Zugriff durch Nutzer auf das vertriebene Produkt und die Möglichkeiten zum Markteintritt durch andere Anbieter sind demnach sehr einfach gegeben und schränken den Preisspielraum der Distributoren stark ein.[85]

Aufgrund der begrenzten Möglichkeiten, mit dem Verkauf von Distributionen Erlöse zu erzielen, bieten viele Distributoren ebenfalls Consulting-, Implementierungs- und Trainingsdienstleistungen an. Hierbei ist ihnen ihr technisches Know-how, das sie durch die Zusammenstellung der Software erreicht haben sehr nützlich. Allerdings konkurrieren sie mit etablierten IT-Beratungsunternehmen, die für ihre Kunden die Softwarezusammenstellung und -installation für einen bestimmten Bedarfsfall übernehmen.[86] Dabei sind IT-Beratungsunternehmen nicht auf das Angebot von Open Source-Software beschränkt, sondern können die für ihren Kunden beste Lösung plattformneutral aus kommerzieller oder Open Source-Software auswählen.

2.2.2.2 Verkauf zusätzlicher Software

Geschäftsmodelle, die auf dem Verkauf zusätzlicher Software aufsetzen, nutzen Open Source-Software als Einstieg und Basis für den Verkauf komplementärer kommerzieller Softwareprodukte oder auch für das Angebot von Supportdiensten. Unternehmen, die dieses Business-Modell betreiben, haben oft eine sehr enge Beziehung zu einzelnen Open Source-Projekten. Häufig sind die Unternehmensgründer die Initiatoren der Projekte oder haben wichtige Entwicklerfunktionen übernommen. Die wichtigsten Entwickler der jeweiligen Software können dabei innerhalb der Unternehmen beschäftigt werden. Dennoch sind auch diese Projekte auf die Unterstützung der Entwicklergemeinde angewiesen. Je kommerzieller die Ausrichtung dieser Unternehmen ist, desto schwieriger wird es jedoch, genügend freiwillige Entwickler zu gewinnen, denn diese steht den auf Freiwilligkeit und kostenlose Verfügbarkeit ausgerichteten Entwicklerinteressen entgegen. Dies zeigt auch das bereits angeführte Beispiel Netscape.[87]

Basis dieser Geschäftsmodelle ist eine kostenlos erhältliche Open Source-Software auf der zusätzliche add-ons oder Programme mit erweitertem Funktionsumfang aufbauen, die dann kostenpflichtig vertrieben werden. Ferner ist es möglich, im Rahmen einer Dual-Licensing Strategie ein Produkt als Open Source-Software frei

[85] Vgl. EVANS UND REDDY (2002), S. 37.

[86] Vgl. BERLECON RESEARCH (2002c), S.43.

[87] Vgl. 2 Abschnitt 2.1.3.

zugänglich zu machen und dasselbe Produkt als kostenpflichtige Version an Unternehmen zu verkaufen, die es zusammen mit anderer kommerzieller Software einsetzen wollen.

So bietet etwa MySQL seine Datenbank unentgeltlich als Open Source-Software an. Es gibt aber auch die Möglichkeit, die Software mit einer speziellen Unternehmenslizenz zu erwerben, so dass sie zusammen mit anderer kommerzieller Software eingesetzt werden kann.[88] Ximian bietet eine im Aussehen und Funktionsumfang stark an Microsoft Outlook angelehnte Software an. Als add-on zu dieser Software kann man eine kostenpflichtige Komponente erwerben, die die Zusammenarbeit mit einem bestimmten E-Mail-Server ermöglicht.

Ein Verkauf zusätzlicher Software ist aber auch durch kommerzielle Softwareanbieter wie Oracle, SAP oder IBM möglich. Diese Anbieter haben in der Regel mit bestimmten Softwareprodukten eine bedeutende Marktposition auf dem kommerziellen Softwaremarkt erreicht. Die Anpassung ihrer Software an andere Plattformen erlaubt es ihnen, die mögliche Installationsbreite zu vergrößern. Dabei ist von Bedeutung, dass die Open Source-Plattform als kostenlose Basis genutzt werden kann. Ist es für den Nutzer der Software nicht notwendig, zusätzlich in eine kommerzielle Plattform zu investieren, so sind die Erlösmöglichkeiten auf der Ebene der Applikationen entsprechend größer. Wichtig zu unterscheiden ist dabei, dass die von kommerziellen Softwareanbietern für eine Open Source-Plattform angebotene Software selbst keine Open Source-Software sein muss.[89] Diese Software ist lediglich Open Source-kompatibel und nutzt Open Source als Betriebssystemplattform.[90]

2.2.2.3 Verkauf von zusätzlicher Hardware

Unternehmen können auch Open Source-Projekte unterstützen, um den Verkauf ihrer eigenen Hardwareprodukte zu fördern. IBM ist das bekannteste Beispiel für eine solche Strategie. Dabei ist IBM nicht nur Hardwarehersteller, sondern gleichzeitig auch einer der größten Softwarehersteller weltweit. Nach eigenen Angaben hat IBM bis zum Jahre 2000 1 Mrd. US$ in verschiedene Open Source-Projekte in-

[88] Siehe www.mysql.com.

[89] Die GPL greift nur dann, wenn neue Software Bestandteile von anderer GPL-lizensierter Software verwendet („viral effect").

[90] Siehe zu einem ökonomischen Modell zur Unterstützung von Open Source-Software durch kommerzielle Anbieter Mustonen (2002).

vestiert. Hierzu gehört die Anpassung von Linux und Apache an die verschiedenen IBM-Hardwareplattformen.[91]

IBM nutzt angepasste Linuxversionen, um für verschiedene Serverplattformen eine einheitliche Betriebssystembasis zu schaffen, auf der dann die verschiedenen IBM Softwarekomponenten ausführbar sind. Die Software wird somit stark in die Hardware integriert und ermöglicht eine Homogenisierung der verschiedenen Hardwareprodukte. Wird die Betriebssystemplattform als ein fester Bestandteil der Hardware angesehen, können Erlöse dann mit dem Verkauf der Hardware erzielt werden. Dies ist für Hardwarehersteller insbesondere deshalb interessant, weil für den Nutzer keine zusätzlichen Kosten für den Erwerb eines Betriebssystems anfallen. Wenn der Nutzer weniger Softwareausgaben hat, erhöht sich aber auch der Erlösspielraum für den Hardwarehersteller.

Open Source-Projekte werden ebenfalls von Intel unterstützt. Dabei steht das Interesse im Vordergrund, Software zu schaffen, die optimal mit den Intel Hardwareprodukten harmoniert. Von daher kann die Unterstützung eher als Verkaufsförderung für die eigenen Produkte angesehen werden denn als eigenständiges Geschäftsfeld.

2.2.2.4 Bewertung der Geschäftsmodelle und limitierende Faktoren

Allen kommerziellen Open Source-Geschäftsmodellen ist gemeinsam, dass sie nach marktwirtschaftlichen Kriterien funktionieren müssen. Unternehmen investieren nur dann in die Entwicklung von Open Source-Software, wenn sie zusätzliche Dienste oder Produkte anbieten können, mit denen sich die Investitionen finanzieren lassen. Open Source wird in diesen Geschäftsmodellen zu einem günstigen, weil kostenlosen Inputfaktor, auf dem mit aufbauenden Diensten versucht wird, zusätzliche Erlöse zu generieren. Die Basis dieser Geschäftsmodelle bleibt dabei jedoch die frei zugängliche Software. Dies limitiert die Möglichkeiten, stabile Erlösmodelle aufzubauen und begrenzt auch die Anreize, in Open Source-Projekte zu investieren.

Im Falle der Beratungs-, Trainings- und Supportangebote unterscheiden sich die Open Source-Geschäftsmodelle nicht von Geschäftsmodellen in der kommerziellen Softwarewelt. Gleiches gilt für den Vertrieb von spezieller Literatur oder das Angebot von Messen und Zeitschriften.

[91] Vgl. EVANS UND REDDY (2002), S. 34.

Wichtig für den Erfolg der kommerziellen Open Source-Geschäftsmodelle ist, dass das Zusatzangebot nicht auch leicht von einem anderen Unternehmen angeboten werden kann – was voraussetzt, dass Eigentumsrechte an diesem Zusatzangebot durchgesetzt werden können. Das Beispiel der Distributoren hat gezeigt, dass hier die Markteintrittschranken für andere Unternehmen sehr gering sind. Gleiche Softwarepakete oder Supportangebote können auf der Basis der von den Distributoren geleisteten Arbeit auch von anderen Unternehmen angeboten werden. Wenn die eigentliche Erlösquelle für Open Source-Projekte aber Zusatzprodukte sind, an denen die Unternehmen Eigentumsrechte durchsetzen können, ist dies eher ein Beleg für die Überlegenheit herkömmlicher kommerzieller Geschäftsmodelle als für die Stabilität von kommerziellen Open Source-Geschäftsmodellen. Kommerzielle Open Source-Geschäftsmodelle scheinen besonders dann erfolgreich zu sein, wenn sie möglichst weit von der Open Source-Welt entfernt sind und die Eigentumsrechte an den Zusatzleistungen gut durchgesetzt werden können. Dies bedeutet, dass für den Anbieter einer komplementären Leistung eine exklusive Vermarktung seines Angebotes möglich ist. Es komnmt dann zu keinem Preisdruck, weil andere Anbieter das entsprechende Angebot nicht leicht kopieren können. Damit unterscheiden sich erfolgreiche kommerzielle Open Source-Geschäftsmodelle aber auch nicht von herkömmlichen kommerziellen Geschäftsmodellen.

Im Fall spezieller Lizenzen für kommerzielle Versionen oder Komponenten von Open Source-Software können die Zusatzkomponenten exklusiv vermarktet werden. Auch Hardwarehersteller können Open Source-Software als einfache Möglichkeit auffassen, ihre Hardware mit einer höheren Funktionalität auszustatten. Aufgrund der sehr guten exklusiven Vermarktungsmöglichkeiten von Hardwareprodukten erscheint hier die Integration von Open Source vergleichsweise einfach.

Verschiedene Faktoren begrenzen die Möglichkeiten innerhalb eines kommerziellen Open Source-Geschäftsmodells. Je kommerzieller Open Source-Geschäftsmodelle zusätzliche Produkte vermarkten, desto schwieriger könnte es sein, genügend freiwillige (aus Unternehmenssicht kostenlose) Unterstützung durch die Entwicklergemeinde für ein Projekt zu erhalten. Damit steigt aber die Notwendigkeit, in den Aufbau einer eigenen fest angestellten Entwicklungabteilung zu investieren. Die Arbeit dieser Entwickler wäre aber auch allen anderen Nutzern frei zugänglich. Je größer die eigenen Entwicklungsinvestitionen jedoch sind, desto eher bestehen Anreize, diese Investitionen auch zu schützen und die freie Zugänglichkeit z. B. durch spezielle Lizenzmodelle einzuschränken. Damit entfernen sich diese Modelle mit

zunehmender Kommerzialisierung immer stärker von der Open Source-Welt und gleichen sich herkömmlichen Geschäftsmodellen an.

Ein weiterer begrenzender Faktor ist die geringere Differenzierungsmöglichkeit für ein Unternehmen innerhalb der Open Source-Entwicklungsumgebung. Da die Ergebnisse der Programmiertätigkeit allen anderen potenziellen und tatsächlichen Anbietern ebenfalls zur Verfügung stehen, sind nicht nur sehr leicht Kopien des Produktes, sondern auch der Geschäftsstrategie möglich. Höhere Erlösmöglichkeiten bestehen damit für den ersten Anbieter eines bestimmten Produktes nicht sehr lange. Basiert ein Geschäftsmodell auf einem Open Source-Produkt, so ist dessen Marktposition von Imitatoren sehr leicht zu bestreiten (schnelle Erosion von Pioniergewinnen).

2.3 Konsequenzen aus dem Fehlen marktlicher Koordination im Open Source-Modell

2.3.1 War Software jemals frei? Die Entstehung des Softwaremarktes aus ökonomischer Sicht

In Darstellungen der Open Source-Entwicklungsgeschichte wird häufig darauf verwiesen, dass am Anfang alle Software frei war. Damit wird Bezug genommen auf die bereits dargestellte Form der Softwareentwicklung zu Beginn der Verbreitung von Computern. Spezialisten und Systemadministratoren entwickelten zu dieser Zeit Software für Großrechner und stellten die eigenen Veränderungen und Weiterentwicklungen wieder anderen Nutzern zur Verfügung. Die Entwickler selbst waren an Universitäten oder bei Großunternehmen angestellt und wurden von diesen bezahlt. Da der Abnehmerkreis für spezielle Softwareentwicklungen sehr klein war, hatten die Arbeitgeber der Entwickler zunächst selten ein Interesse, die Software extern zu vermarkten. Großrechner und die benötigte Software wurden primär als Arbeitsmittel und Inputfaktor und nicht als vermarktbares Produkt angesehen.

Entscheidend für die Bereitschaft, eigene Softwareentwicklungen auch für andere Nutzer zu öffnen und die Software kostenlos weiterzugeben, war die weitgehende Deckungsgleichheit von Entwicklern und Nutzern. Aufgrund der nur sehr geringen Verbreitung von Großrechnern hatte jeder Entwickler einen Anreiz, seinen Entwicklungsbeitrag für andere Nutzer zu öffnen, da er davon ausgehen konnte, dass diese anderen Nutzer ebenfalls die Software weiterentwickeln werden. Damit war ein Tauschhandel der Entwicklungsbeiträge möglich. Solange es keine oder nur wenige

Nutzer gab, die nicht auch selbst Entwicklungsbeiträge leisteten, bestand nicht die Gefahr, dass „Trittbrettfahrer" nur von der Softwareentwicklung profitierten, selbst aber nicht zur Weiterentwicklung beitrugen. Damit war Software auch zu Beginn der Softwareentwicklung nicht marktfrei, sie wurde lediglich im Tauschhandel verbreitet. Der eigene Programmierbeitrag wird dabei gegen den Programmierbeitrag eines anderen Entwicklers getauscht.

In der Darstellung der Kollektivgütertheorie kann die Entwicklung von Software zu dieser Zeit als Klubkollektivgut eingeordnet werden. Es liegt keine Rivalität im Konsum vor und eine Exkludierbarkeit wird indirekt dadurch gewährleistet, dass nur derjenige Software nutzt, der diese ebenfalls weiterentwickeln kann. Damit entsteht indirekt ein Klub, dem nur die Nutzer angehören können, die auch Entwickler sind.

Die Deckungsgleichheit von Softwareentwicklern, die gleichzeitig Anwender sind, zerfällt mit der weiteren Verbreitung von Computern. Zunächst entsteht Software noch als Individualsoftware bzw. in geringer reproduzierter Auflage für spezielle Geschäftsanwendungen. Schon hierbei leistet der Nutzer der Software als Gegenleistung keinen eigenen Entwicklungsbeitrag, sondern bezahlt für die Nutzung der Software.

Mit der Entstehung eines Massenmarktes für Personal Computer gibt es schließlich endgültig eine sehr große Zahl an Nutzern, die selbst keinen eigenen Entwicklungsbeitrag leisten. Der implizite Klub der Entwickler-Nutzer zerfällt. Ohne Festlegung von Eigentumsrechten ist die Bereitschaft gering, einen Entwicklungsbeitrag zu leisten, der dann von vielen unentgeltlich genutzt werden kann. Software kann in diesem Zwischenschritt als reines Kollektivgut angesehen werden, für das es keine Rivalität und auch keine Exkludierbarkeit gibt. Erst die Definition von Eigentumsrechten schafft wieder eine Exkludierbarkeit und damit Anreize, in die Entwicklung von Software zu investieren.

Auch bei der Entstehung der Free Software Foundation wird die Gefahr gesehen, dass frei entwickelte Software ohne eigenen Entwicklungsbeitrag genutzt wird. Die Nutzung bezieht sich hierbei jedoch auf andere Untenehmen, die freie Software in ihre kommerziellen Produkte integrieren könnten, ohne diese selbst zu öffnen. Die Free Software Foundation spricht im Zusammenhang mit der Entwicklung freier Software selbst von einem „Klub", dem nur derjenige beitreten und die entwickelte

Software weiterverwenden kann, der seine Software ebenfalls öffnet.[92] Die Ausgestaltung der GPL soll genau dies ermöglichen. Damit ist gewährleistet, dass niemand den Softwarecode für seine Softwareprodukte verwendet, der nicht selbst seinen Entwicklungsbeitrag für andere öffnet. Die GPL wirkt somit als Exklusionsmöglichkeit für diejenigen Unternehmen, die ihre Software nicht öffnen.

2.3.2 Ökonomische Motive zur Partizipation an Open Source-Projekten

Hinsichtlich der Motivation von Open Source-Entwicklern wird teilweise argumentiert, dass es sich innerhalb der Open Source-Bewegung um eine „gift culture" handelt, die von Altruismus und Gegenseitigkeit motiviert ist. Danach tragen Entwickler zu Open Source-Projekten bei, weil sie Spaß daran haben, Teil der Community zu sein und den eigenen Entwicklungsbeitrag als Gegenleistung für Programme und Hilfestellungen sehen, von denen sie selbst profitiert haben.[93] Der Tausch „Ware gegen Geld" soll dabei abgelöst werden durch einen Tausch „Geschenk gegen Reputation".[94] Fraglich bleibt dabei jedoch, warum gerade bei der Erstellung von Software die Motive Altruismus und Gegenseitigkeit eine größere Bedeutung haben sollen als in anderen Sektoren. Das Motiv einer altruistischen Motivation scheint vor allem dann als Erklärung ungeeignet, wenn nicht andere Privatanwender von der Programmierarbeit profitieren, sondern Großunternehmen. Auf Gegenseitigkeit beruhende Austauschbeziehungen spielen vor allem in kleinen, überschaubaren Gruppen eine Rolle.[95] Es ist dagegen sehr fraglich, ob Gegenseitigkeit auch als Erklärung für den Programmierbeitrag in einer sehr großen anonymen Gruppe dienen kann.[96]

Zur Untersuchung der Entwicklermotive hat die Boston Consulting Group im Jahre 2002 eine Befragung der bei SourceForge.org registrierten Entwickler durchgeführt. Dabei haben die Befragten als Motive die in der Abbildung 8 dargestellten Begründungen für ihre Arbeit angegeben.[97]

---

[92] Vgl. FREE SOFTWARE FOUNDATION „The GNU GPL and the American Way" www.gnu.org/philosophy/gpl-american-way.html

[93] Vgl. RAYMOND (1998b) und SCHMIDT UND SCHNITZER (2003), S. 10.

[94] Vgl. RAYMOND (1999), KOLLOCK (1999) und FRANCK UND JUNGWIRTH (2002), S. 124.

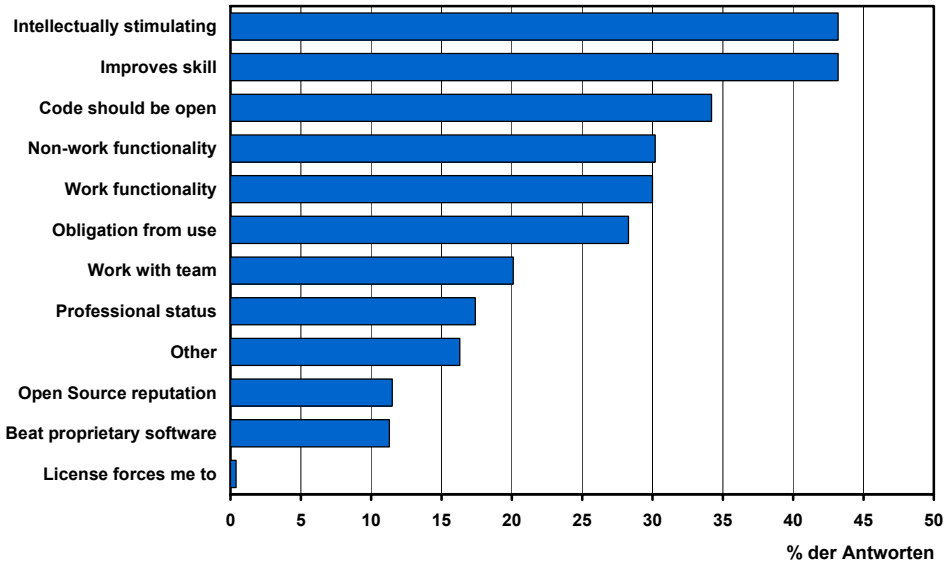
[95] FEHR UND SCHMIDT (2002) geben einen Überblick über die ökonomische Forschung zum Thema Fairness und Reziprozität.

[96] Vgl. SCHMIDT UND SCHNITZER (2002), S. 10. LERNER UND TIROLE (2000), S. 18.

[97] Vgl. BOSTON CONSULTING GROUP (2002), S. 16.

Motive wie „Intellectually stimulating", „Work with team" oder „Non-work functionality" können als Motive angesehen werden, wie sie auch für andere Tätigkeiten zur Freizeitgestaltung gelten. Auch ideologische Motive wie „Code should be open" finden sich in ähnlicher Form bei anderen Freizeitaktivitäten. Damit kann die Open Source-Entwicklung als normale Form der Freizeitbeschäftigung angesehen werden, deren Ertrag auch anderen Nutzern zur Verfügung steht.[98]

Abbildung 8: Motive von Open Source-Entwicklern



Quelle: BOSTON CONSULTING GROUP (2002), S. 16.

Daneben finden sich aber auch zahlreiche Motive, die in direktem oder indirektem Zusammenhang mit einer beruflichen Tätigkeit stehen. Hierzu gehören etwa „Improve skill", „Work functionality", „Professional status" oder „Open Source reputation". Im Folgenden sollen diese ökonomischen Motive zur Partizipation an einem Open Source-Projekt genauer untersucht werden. Dabei wird unterschieden nach der Bedeutung von Open Source zur Offenlegung der eigenen Entwicklerqualitäten, dem Leisten von Kleinbeiträgen und der Nutzung von Open Source zur Weiterbildung.

Nach ökonomischen Kriterien partizipiert ein Entwickler nur dann an einem Projekt, wenn mit dieser Aktivität im Vergleich zu seinen Kosten ein positiver Nutzen

[98] Ein ähnlicher Effekt tritt bei Kleingartenvereinen auf, deren Anlagen auch für die Öffentlichkeit geöffnet sind. Auch hier ergibt sich durch die Freizeitbeschäftigung der Kleingärtner ein positiver Nutzen

für ihn verbunden ist. Kosten entstehen ihm durch die in die Programmiertätigkeit investierte Zeit. Wie hoch er diese Kosten bewertet, hängt davon ab, wie viel Spaß ihm die Arbeit bereitet. Ein Nutzen kann ihm durch eine direkte oder spätere Gegenleistung entstehen.[99]

In einem kommerziellen Softwareprojekt ist die Bezahlung die direkte Gegenleistung für den Entwickler. Eine direkte Gegenleistung kann für den Entwickler auch darin bestehen, dass ein ihn persönlich betreffendes Software-Problem direkt gelöst wird. Dies kann z. B. die Bereitstellung eines bestimmten Treibers sein, der dann gemeinsam entwickelt wird, die Anpassung der Software an seinen individuellen Bedarf oder auch die Behebung eines Programmfehlers.

Leistet ein Entwickler nur **kleine Entwicklungsbeiträge** innerhalb eines Open Source-Projektes, so sind diese für einen hochqualifizierten Entwickler nur mit wenig Aufwand und geringen Kosten verbunden. Hierzu gehört es etwa, ein einfaches Problem zu lösen, eine Software an die individuellen Bedürfnisse anzupassen oder eine kleine Zusatzapplikation zu entwickeln. Wenn diese Entwickler Zugang zum Source Code haben, ist es für sie relativ einfach, Verbesserungen vorzunehmen von denen auch andere in ihrer täglichen Arbeit profitieren. Die Opportunitätskosten, den neuen Code mit anderen zu teilen, sind ebenfalls gering. Sind die Neuentwicklungen zudem relativ anspruchslos, lohnt es sich nicht, diese Erfindung schützen zu lassen. Außerdem offeriert das Internet einen effizienten und kostengünstigen Weg, die Innovation der Öffentlichkeit zugänglich zu machen. Für den einzelnen Entwickler sind damit die Kosten, einen Entwicklungsbeitrag zu leisten und offen zu legen, sehr gering.[100]

Des Weiteren kann die Open Source-Entwicklungsarbeit auch zur **Weiterbildung** genutzt werden. Hierbei nutzt der Entwickler die Mitarbeit an einem Open Source-Projekt, um Fragestellungen aus seinem Berufsalltag zu lösen. Für seinen Entwicklungsbeitrag erhält er somit in Form der Problemlösung eine direkte Gegenleistung. Die Programmierarbeit zur Weiterbildung kann aber als Investition in seine berufliche Zukunft angesehen werden. Hierbei würde die zukünftige Gegenleistung in Form einer möglichen besseren Berufsperspektive bestehen.

---

für die Besucher.

[99] Siehe zu den Kosten-Nutzen-Überlegungen LERNER UND TIROLE (2000), S. 14 und MENDYS-KAMPHORST (2002), S. 19-20

[100] Vgl. SCHMIDT UND SCHNITZER (2003), S. 10-11.

Eine zukünftige Gegenleistung kann auch darin bestehen, dass Entwickler über die Programmierarbeit ihre **Qualitäten als Programmierer offen legen** (Signaling) und einer Bewertung unterziehen können. Die Arbeit in einem Open Source-Projekt produziert somit Signale über die Qualität der eigenen Arbeit, die dann auf einem Sekundärmarkt (Arbeitsmarkt) gewinnbringend verwertet werden können. Die Entwicklungsarbeit wird somit zum Signaling und zum Aufbau von Reputation genutzt.[101] Auch bei wissenschaftlichen Veröffentlichungen ist ein analoges Verhalten festzustellen.[102]

Die Offenlegung der eigenen Entwicklerqualitäten und der Aufbau von Reputation können ein wichtiges Motiv sein, wenn die Entwicklungsarbeit über einfache Kleinbeiträge hinausgehen soll. Bei größeren Projekten wird wie bereits beschrieben der Hauptteil der Entwicklungsarbeit von einer kleineren Gruppe von Entwicklern geleistet. Mit einem maßgeblichen Entwicklungsbeitrag können diese Entwickler ihre Reputation erhöhen und diese in Bereichen außerhalb der Entwicklung verwerten. Ist das Motiv des Reputationsaufbaus ein wichtiges Argument für einen Entwicklungsbeitrag, dann ist der Anreiz umso größer, an einem Open Source-Projekt teilzunehmen, desto größer der kommerzielle Markt ist, auf dem die in der Open Source-Entwicklung erlangte Reputation verwertet werden kann. Gibt es einen großen kommerziellen Softwaremarkt, so ist die Anzahl der Unternehmen größer, die Entwickler für ihre kommerzielle Entwicklungsarbeit entlohnen würden.[103] Verschiedene Faktoren haben Einfluss auf die Größe des Signalisierungseffektes:

- Der Signalisierungseffekt ist dann groß, wenn die eigene Leistung von einem großen Publikum bewertet werden kann. Dies kann dazu führen, dass Entwickler solche Projekte bevorzugen, die möglichst viele andere Programmier anziehen. Große Projekte, an denen bereits viele Entwickler beteiligt sind, sind demnach attraktiver als kleine Nischenprojekte, deren weiterer Verlauf und deren Bedeutung noch ungewiss sind. Als positiver Effekt hieraus ergibt sich, dass in dem Feld, das einer großen Aufmerksamkeit unterliegt, tendenziell qualitativ hochwertigere Arbeit geleistet wird. Hinsichtlich der Ressourcenallokation konzentriert

[101] Vgl. FRANCK UND JUNGWIRTH (2002), S. 127.

[102] Siehe LEE, MOISA UND WEISS (2003), S. 23 für einen Vergleich mit den Anreizen in der Wissenschaft. Auch hier dienen wissenschaftliche Publikationen dem Aufbau von Reputationen, um Fähigkeiten zu signalisieren und somit künftige berufliche Perspektiven zu verbessern.

[103] Vgl. MENDYS-KAMPHORST (2002), S. 20. Die Empirie scheint diese These zu belegen: *„contributing to open source did help many programmers to get access to venture capital or to be offered attractive jobs by commercial software developers*". EBENDA (2003), S.12.

sich jedoch alle Entwicklerkapazität in wenigen Feldern, während andere Bereiche unbearbeitet bleiben (Vgl. zur Ressourcenallokation 2.3.5.2).[104]

- Der Signalisierungseffekt wird auch dann stärker sein, wenn die zu leistende Programmieraufgabe besonders anspruchsvoll ist und die Gruppe der Entwickler in der Lage ist, den Entwicklungsbeitrag einzuschätzen und zu bewerten. Er ist besonders bedeutsam in *„settings with sophisticated users and an audience that can appreciate effort and artistry and thus distinguish between merely good and excellent solutions to problems*".[105] Ein wichtiger Aspekt in diesem Zusammenhang ist das sog. „peer review". Dies bedeutet, dass die Arbeit eines Programmierers durch andere fachlich kompetente Programmierer überprüft werden kann. Nur mir Hilfe von „peer review" kann eine gewisse Reputation aufgebaut werden, da dies das einzige Maß für die Qualität eines Produktes darstellt. Die Anreize, die zum Signaling führen, können somit Entwicklungskapazitäten in Bereiche lenken, die nicht den tatsächlich benötigten Produkten entsprechen müssen.

In der praktischen Ausgestaltung von Open Source-Projekten lassen sich Anzeichen finden, die auf die Bedeutung des Signalings hinweisen:[106]

- Die Angabe des Entwicklungsbeitrags ist in einem Projekt von besonderer Bedeutung.[107] In der project history, den credits oder der maintainer list werden die jeweiligen beteiligten Entwickler aufgeführt. Auch viele Webseiten von Open Source-Projekten listen die beteiligten Entwickler die Bedeutung ihrer Beiträge auf. Die Webseite des Webservers Apache etwa zeigt sehr genau die beteiligten Entwickler und ihre jeweiligen Beiträge.[108] Auch auf der Webseite von Sourceforge werden die registrierten Projekte und die Entwickler mit ihrer Qualifikation aufgeführt und ihre Beiträge in einem Ranking dargestellt.[109] Ebenso wird es auch als schweres Vergehen angesehen, wenn der Beitrag eines Entwicklers nicht mehr zu erkennen ist. *„removing a person's name from a project history,*

[104] LERNER UND TIROLE (2002), S.15-16 weisen darauf hin, dass auch in der wissenschaftlichen Forschung ähnliche Entwicklungen zu finden sind. Während auf einigen Feldern sehr intensive Forschung stattfindet, bleiben andere Gebiete vollständig unbearbeitet.

[105] Vgl. WEBER (2000), S. 22.

[106] Vgl. SCHMIDT UND SCHNITZER (2003), S. 11.

[107] Vgl. MENDYS-KAMPHORST (2002), S. 13.

[108] Vgl. www.apache.org

[109] Vgl. sourceforge.net

credits or maintainer list is absolutely no done without the person's explicit consent. … surreptitiously filing someone's name off a project is, in cultural context, one of the ultimate crimes."[110]

- Die ersten Open Source-Projekte waren insb. technisch anspruchsvolle Lösungen für die Betriebssystemebene. Solche Lösungen sind am ehesten geeignet, Reputation aufzubauen und bieten damit einen höheren Signalisierungsanreiz als Aufgaben, die eher den weniger fortgeschrittenen Nutzern zugute kommen. [111]
- Der modulare Aufbau vieler Projekte erleichtert nicht nur die verteilte Zusammenarbeit in verschiedenen Entwicklerteams, er ermöglicht auch eine genauere Zuordnung der Einzelbeiträge.[112]

Die Untersuchung der Motive hat bezeigt, dass es auch zahlreiche ökonomische Erklärungen für die Entwicklerpartizipation an einem Open Source-Projekt gibt, weil Entwickler dadurch auch ihre eigene berufliche Situation verbessern können. Hierzu ist jedoch die Existenz eines kommerziellen Softwaremarktes zwingend erforderlich. Verfolgt der Entwickler seine Eigeninteressen, so muss dies jedoch nicht zum besten Ergebnis für den Nutzer der Software führen. Wie gezeigt, werden die Entwicklerinteressen dann am besten erfüllt, wenn technisch anspruchsvolle Lösungen entworfen werden. Dies sind jedoch in der Regel andere Ansprüche als Endnutzer an eine Software stellen.[113]

2.3.3 Die Open Source-Softwareentwicklung – Kein Bazar

In einem der bekanntesten Beiträge über die Open Source-Entwicklungsprinzipien wird die Arbeit an einem Open Source-Projekt mit einem „Bazar" verglichen: „*The Linux community seems to resemble a great babbling bazaar of differing agendas and approaches… out of which a coherent and stable system could seemingly emerge only by a succession of miracles*."[114] Demgegenüber wird die Entwicklungsarbeit an einem kommerziellen Projekt als „Kathedralenbau" und damit als streng hierarchische Organisationsform angesehen.

---

[110] Vgl. RAYMOND (1998b)

[111] Vgl. WEBER (2000), S.21f.

[112] Vgl. LERNER UND TIROLE (2000), S. 17.

[113] Siehe zu einer modelltheoretischen Analyse der Entstehung von Open Source-Software und der Entwicklerbeiträge auch JOHNSON (2001).

[114] Vgl. Raymond „The cathedral and the bazaar."

Eignet sich das Bazar-Beispiel, um die vielfältige Zusammenarbeit zu verdeutlichen, so fehlt einem Open Source-Projekt jedoch das für einen Bazar bestimmende Merkmal: Das Aufeinandertreffen von Anbietern und Nachfragern und die sich daraus ableitende Preisbildung für das gehandelte Gut. Ein Bazar ist der Prototyp eines Marktes.[115] Das scheinbar chaotische Handeln auf einem Markt ist Ausdruck eines dezentralen Koordinationsmechanimus, der die Nachfragerwünsche und Anbietermöglichkeiten zusammenbringt. Der Bazar dient dem Informationsaustausch und der Preisbildung. Preise zeigen dabei die relativen Knappheiten an und sind eine wichtige Information für Anbieter, wie die Nachfrager den Wert einer Ware einschätzen und welche Waren am stärksten nachgefragt werden. Jeder Händler verfolgt auf einem Bazar seine Eigeninteressen, indem er seinen Gewinn maximiert. Dies ist ihm dann am besten möglich, wenn er die Waren anbietet, die von den Kunden am besonders dringlich gewünscht werden. Der Händler, der die Kundenwünsche am besten erfüllt, hat die größten Gewinnaussichten. Damit profitiert der Nachfrager indirekt von den Eigeninteressen des Anbieters.[116]

In einem Open Source-Projekt treffen keine Anbieter und Nachfrager aufeinander. Der Open Source-Bazar besteht nur aus Anbietern bzw. Entwicklern, die ihre Software primär nach eigenen Vorstellungen entwickeln. Zwar verfolgen die Entwickler, wie im vorangehenden Abschnitt gezeigt, auch ihre Eigeninteressen. Es gibt dabei jedoch keine Rückkoppelung zu den Interessen der Nachfrager und Nutzer einer Software. Die Software, die bei Verfolgung der Entwicklerinteressen entsteht, muss nicht die Software sein, die die Kunden wünschen.

2.3.4 Bedeutung des Marktes und Erfüllung der Marktfunktionen bei Open Source

In einer Marktwirtschaft erfolgt die Abstimmung von Angebot und Nachfrage über den Markt im Wettbewerb verschiedener Anbieter untereinander. Im Gegensatz zu einer zentralen Planwirtschaft ist der Markt ein dezentrales Planungs- und Koordinationsinstrument, d.h. es gibt keine gesamtwirtschaftlichen Vorgaben hinsichtlich der Produktionsmenge oder Rationierungen und Festlegungen der nachzufragenden

[115] Zur konstituierenden Rolle der Preisbildung für einen Markt siehe BORCHERT, GROSSEKETTLER (1985), S. 13 ff.

[116] Vgl. SMITH (1776), Book 1, Chapter2, der diesen Mechanismus bereits vor über 200 Jahren aufgezeigt hat: „*It is not from the benevolence of the butcher, the brewer, or the baker that we expect our dinner, but from their regard to their own interest. We address ourselves, not to their humanity but to their self-love, and never talk to them of our own necessities but of their advantages*".

Mengen. Wie im vorangehenden Teil dargestellt, sind es die Eigeninteressen der Anbieter, die sie dazu bringen, die Interessen der Nachfrager am Besten zu befriedigen.

Gibt es auf einem Markt Knappheiten, so muss ein Rationierungsverfahren gefunden werden, das regelt, wie der Wettbewerb um knappe Güter ausgetragen wird. Knappheiten können dabei auf dem Endkundenmarkt aber auch beim Einsatz von Produktionsfaktoren vorliegen. Es sind verschiedene Verhaltensweisen und Regelungen denkbar, mit denen Nachfrager die Verteilung knapper Güter organisieren und die eigene Versorgung mit diesen Gütern beeinflussen: Anstehen in Warteschlangen, die Ausübung oder Androhung von Gewalt, Bestechung oder Betrug können in diesem Sinne ebenso Wettbewerbsverfahren sein wie der Preis- und Qualitätswettbewerb im Markt. Die Funktionsweisen dieser Verfahren sind dabei jedoch höchst unterschiedlich. Demnach besteht auch nicht die Wahl, ob man in einer Gesellschaft ohne oder mit Wettbewerb leben kann, da es in einer Welt der Knappheit keine wettbewerbslosen Gesellschaften geben kann. Wählbar sind jedoch die Wettbewerbsverfahren, d.h. die Regeln, nach denen der Wettbewerb ausgetragen wird.[117]

Mit der Entscheidung für den Markt als Koordinationsinstrument ist auch die Entscheidung für ein dezentrales Wirtschaftssystem gefallen, das den Akteuren weitgehende Planungs-, Handlungs- und Wahlfreiheiten einräumt. Märkte stellen hierbei keinen Selbstzweck dar, sondern dienen der Erfüllung verschiedener Funktionen, die in Tabelle 4 zusammenfassend dargestellt sind:

- Das Prinzip der **Konsumentensouveränität** verlangt, dass Angebot und Nachfrage entsprechend den Nachfragerwünschen zum Ausgleich gebracht werden. Die Abstimmung der individuellen Pläne erfolgt über den Preismechanismus und findet auf und zwischen den einzelnen Märkten statt. Durch adäquate Preisvariationen werden Angebotsüberhänge (Verschwendung) und Nachfrageüberhänge (Mangel) abgebaut und so die einzelwirtschaftlichen Pläne der Marktteilnehmer aufeinander abgestimmt. Entscheidend ist hierbei, dass sich die Anbieter nach den Nachfragern richten und nicht umgekehrt: Die letzte Entscheidung über den Erfolg eines Anbieters trifft der Nachfrager durch den Kauf eines Gutes.
- Die **Faktorallokationsfunktion** lenkt knappe Ressourcen in die produktivste

[117] Vgl. VANBERG (1997), S. 707ff.

Verwendung. Ausgehend davon, dass die knappen Produktionsmittel unbegrenzten Nachfragewünschen gegenüberstehen, müssen die knappen Mittel so gelenkt werden, dass sie in die Verwendung gelangen, in der sie am dringlichsten benötigt werden.

- Die **Verteilungsfunktion** des Wettbewerbs führt dazu, dass eine Einkommensverteilung gemäß dem Produktionsbeitrag erfolgt. So erhält derjenige eine höhere Entlohnung, der die Wünsche der Nachfrager besser erfüllt als andere.
- Die **Anpassungsfunktion** soll dafür sorgen, dass das Verhalten der Marktteilnehmer zügig an geänderte Rahmenbedingungen angepasst wird. Kann z. B. ein Anbieter mit seinem Produkt nicht mehr am Markt bestehen, weil sich die Nachfragewünsche geändert haben, so wird er sein Angebot entsprechend anpassen müssen oder aus dem Markt ausscheiden.
- Die Fortschrittsfunktion sorgt für Anreize, neue Produkte und Verfahren zu entwickeln und damit bessere Preis-Leistungskombinationen verfügbar zu machen als die bislang bekannten (Anreizfunktion des Wettbewerbs).

Tabelle 4: Marktfunktionen und die Konsequenzen ihres Fehlens bei Open Source-Software

Marktfunktion	sorgt für	Fehlen bei Open Source-Software führt zu
Konsumenten-souveränität	Ausgleich von Angebot und Nachfrage	Unterversorgung Überversorgung
Faktorallokation	Knappe Ressourcen werden zum dringendsten Bedarf gelenkt	Ressourcenfehlleitung
Verteilung	Einkommensverteilung nach Produktionsbeitrag	Non-Sustainability
Anpassung	Strukturanpassung	---
Fortschritt	Neue Produkte Neue Verfahren	Innovationshemmnisse

.

Innerhalb des wettbewerblichen Koordinations- und Planungsprozesses kommt dem Preissystem eine entscheidende Bedeutung zu. Preise stellen für eine dezentral organisierte Marktwirtschaft, in der zahlreiche autonome Entscheidungsträger aufeinander abgestimmt interagieren müssen, das zentrale Navigationssystem für alle

Beteiligten dar. Dies wird an den verschiedenen Funktionsaspekten eines Preises deutlich:

- **Information**: Der Preis spiegelt Knappheiten wider und ist damit Indikator bezüglich der Nachfragedringlichkeit einerseits (Zahlungsbereitschaft) und des mit dem Verbrauch von Gütern verbundenen Ressourcenverzehrs andererseits (Kosteninformation).
- **Lenkung**: Änderungen auf einer oder beiden Marktseiten führen zu Preisänderungen und dadurch induzierten Mengenreaktionen sowohl auf dem betrachteten Markt selbst als auch auf allen vor- und nachgelagerten Märkten.
- **Motivation**: Der Preis bestimmt das Einkommen des Verkäufers und ist umso höher, je besser Konsumentenwünsche erfüllt werden, woraus sich ein permanenter Innovationsanreiz ergibt.
- **Bewertung**: Nur mit Hilfe von Preisen kann der Erfolg einer ökonomischen Betätigung bewertet und mit anderen Aktivitäten verglichen werden.

Durch die fehlende Bepreisung von Open Source-Software gehen den Entwicklern wichtige Informationen verloren. Für sie liegen keine Informationen über die Zahlungsbereitschaft (= tatsächliche Präferenzen) der Konsumenten vor, aufgrund derer im Marktprozess die Produktionsentscheidungen getroffen würden. Das Fehlen dieser Informationen führt dazu, dass nicht automatisch ein Angebot entsteht, dass sich an den Bedürfnissen der Nutzer orientiert, was sich dann in Überversorgung (Angebotsüberhang) bzw. Unterversorgung (Nachfrageüberhang) äußern kann. Ferner übertragen sich die Funktionsdefizite auf dem Softwaremarkt auf die vorgelagerten Faktormärkte (insbesondere den Arbeitsmarkt für Entwickler) sowie – je nach Finanzierungsmodell der Open Source-Softwareentwicklung – auch auf die nach- bzw. nebengelagerten Komplementärmärkte (z. B. Servicemärkte).

Da das Open Source-Modell im Kern bewusst auf den Markt als Koordinationsmechanismus verzichtet und das Entstehen von Preisinformationen verhindert, können die dargestellten Marktfunktionen vom Open Source-Modell nicht erfüllt werden. Daraus ergibt sich ein systematischer Nachteil der Bereitstellung von Software im Open Source-Modell gegenüber dem kommerziellen Produktionsprozess. Die Tabelle 4 zeigt neben den Marktfunktionen und ihren Aufgaben auch in Stichpunkten die Konsequenzen aus dem Fehlen bei Open Source auf. Die folgenden Gliederungspunkte gehen auf die einzelnen Punkte genauer ein. Dabei werden als Grenzen von

Open Source, die sich direkt aus dem Fehlen des Marktes als Koordinationsinstrument ergeben, die folgenden Punkte genauer behandelt:

- Entwicklerorientierung ist nicht Kundenorientierung
- Schlechtere Lenkung von Ressourcen
- Geringere Innovationsfähigkeit

Darüber hinaus wird die erschwerte Herausbildung von Standards thematisiert, die sich aus den softwarespezifischen Eigenschaften ergibt.

2.3.5 Grenzen von Open Source

2.3.5.1 „Happy Engineering" – Entwicklerorientierung statt Kundenorientierung

Für die Beurteilung eines neuen Produktes ist nicht das technisch maximal Mögliche das entscheidende Kriterium, wichtig ist letztlich die marktliche Verwertbarkeit einer Entwicklung. Haben beispielsweise Ingenieure bei ihren Entwicklungen allein den Technikaspekt im Blick, so entstehen teure, ausgereifte Hochleistungsprodukte, für die sich evtl. kein Käufer findet, weil der Nutzenvorteil für den Endkunden nicht erkennbar bzw. unbezahlbar ist. „Happy Engineering" bezeichnet diese Art der Entwicklung, die die technischen Möglichkeiten ausreizt, ohne nach der Verwertbarkeit, der Bedienbarkeit, dem Nutzengewinn und den dabei auftretenden Kosten zu fragen. Entscheidend für eine Verbreitung von Produkten ist nicht ihr technischer Anspruch, sondern die Schaffung eines Nutzenvorteils für den Kunden.[118] Bei kommerzieller Software stellt der Markt die Kosten der technisch möglichen Lösungen den Kundenwünschen und der sich daraus ableitenden Zahlungsbereitschaft gegenüber. Bei Open Source-Software erfolgt eine solche Gegenüberstellung mangels Bepreisung nicht (vgl. zum Folgenden Abbildung 9 und Abbildung 10).

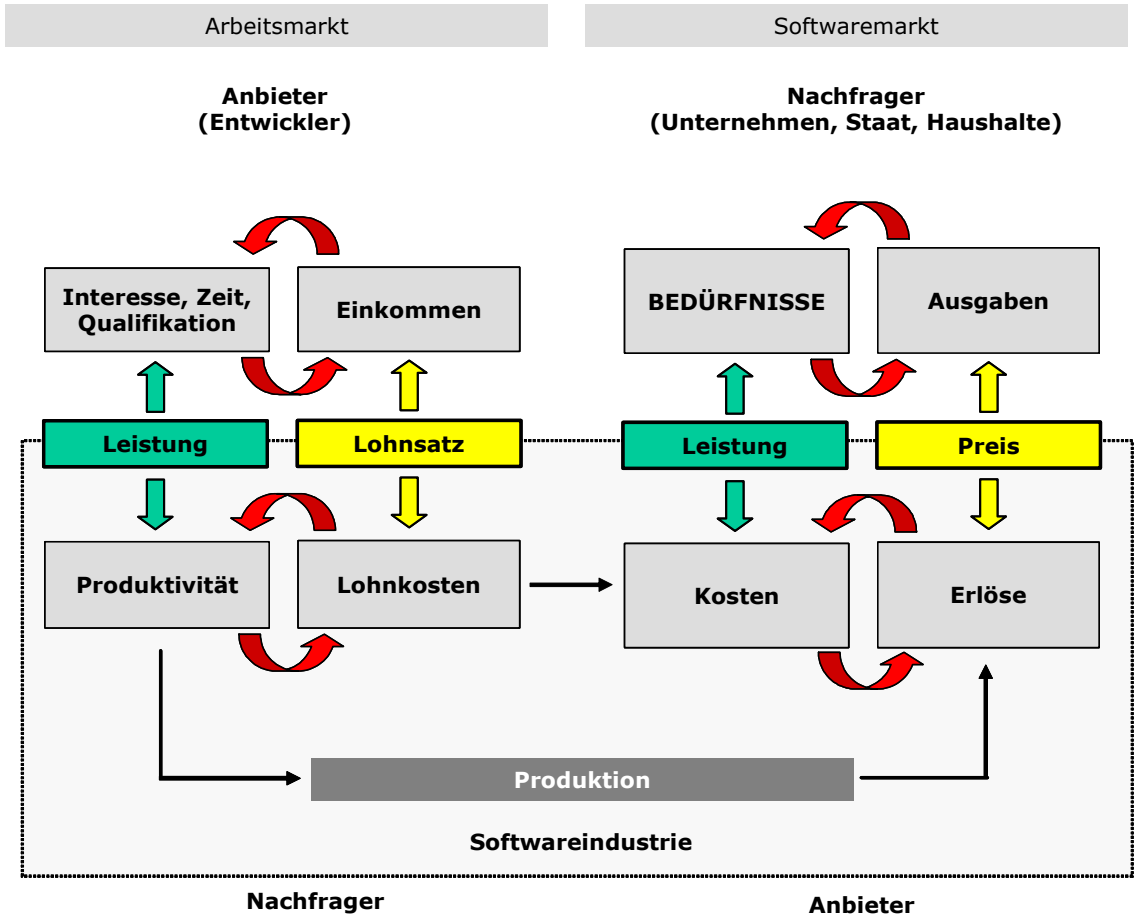
Kommerzielle Software kann auf dem Markt nur dann bestehen, wenn sie die Kundenwünsche bestmöglich erfüllt.[119] Dies muss nicht bedeuten, dass sie immer die technisch aufwendigste Lösung darstellt. Für ein kommerzielles Unternehmen ist der Wert eines Produktes davon abhängig, wie hoch der Nutzen dieses Produktes für den Kunden ist. Diese Wertschätzung seitens des Kunden spiegelt sich in dem

[118] Vgl. BACKHAUS (1999), S. 130-133 der dort auch den Begriff „Happy Engineering" einführt.

[119] Hierzu zählen u. a. Benutzerfreundlichkeit (insbesondere bei Massenmärkten) und Kompatibilität mit bereits installierten Plattformen (Investitionsschutz, Nutzung von Netzwerkeffekten).

Preis wider, den der Unternehmer erzielen kann. Je höher der Nutzengewinn des Kunden, desto höher ist der Erlös des Unternehmens. Daher steht in der traditionellen Softwareentwicklung die Identifikation der Anwenderbedürfnisse und der sich daraus ableitenden Zahlungsbereitschaft am Anfang eines Entwicklungsprozesses. Je wertvoller eine Software aus Sicht der Nachfrager ist (Qualität bzw. Universalität), desto höher sind der Preis bzw. die Absatzmenge und damit die Erlöse, die ein Softwareentwickler für seine Software erwarten kann. Die erzielbaren Erlöse werden bei marktlicher Produktion den Produktionskosten (in Abbildung 9 vereinfachend über die Lohnkosten erfasst) gegenübergestellt. Nur wenn die Nachfrager bereit sind, die bei der Softwareproduktion anfallenden Kosten tatsächlich über den Produktpreis zu finanzieren, ist die Produktion einer Software aus volkswirtschaftlicher Sicht gerechtfertigt. Andernfalls sollten die in der Produktion eingesetzten knappen Ressourcen lieber in andere, produktivere Verwendungen gelenkt werden.

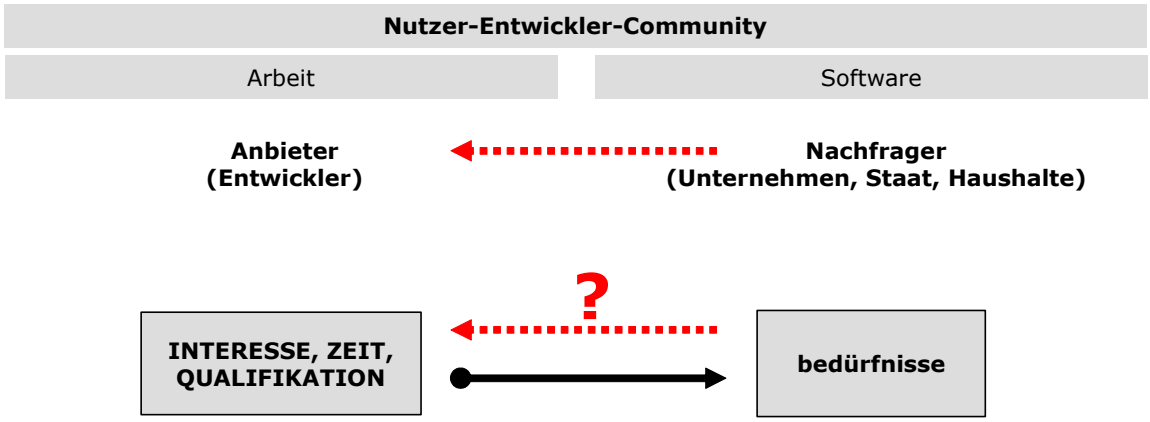
Abbildung 9: Marktverbund bei kommerzieller Softwareproduktion



.

Da kommerzielle Softwareentwickler am meisten verdienen, wenn sie die Software möglichst nahe an den Kundenbedürfnissen entwickeln, ist für sie der Anreiz groß, in Market Research zu investieren und die Kundenbedürfnisse tatsächlich zu identifizieren.[120]

Abbildung 10: Open Source-Softwareproduktion



.

Demgegenüber gibt es bei der Open Source-Softwareentwicklung diese Überprüfung mit den Bedürfnissen einer breiten Nutzerschicht nicht. Die Nutzerbedürfnisse werden daher klein geschrieben – nicht aus Absicht, sondern mangels Information über die tatsächlichen Nachfragerpräferenzen. Der Marktverbund degeneriert zu einer einseitigen Beziehung zwischen Entwicklern und Empfängern. Die Untersuchung der Motive zur Partizipation an einem Open Source-Projekt hat gezeigt, dass Entwickler primär für ihre eigenen Probleme oder Bedürfnisse entwickeln. Dies führt jedoch dazu, dass die Bedürfnisse von Nicht-Entwicklern nur schwer erkannt und umgesetzt werden können. Wenn nicht alle Nutzer Entwickler sind, kann es jedoch dazu kommen, dass Software an den Nutzern (den Kunden, dem Markt) vorbei entwickelt wird.

Der zuvor beschriebene Signalisierungsanreiz wirkt dahingehend, dass ein Programmierer vorrangig daran interessiert ist, statt einer technisch weniger anspruchsvollen Anwendung technisch komplexe Software zu entwickeln, da der Reputationsgewinn bei ersterer vergleichsweise gering ist. Für einen Open Source-

[120] Vgl. SCHMIDT UND SCHNITZER (2003), S. 14.

Entwickler bestehen daher kaum Anreize, Probleme von technisch weniger versierten Anwendern zu identifizieren und zu lösen.[121]

Kundenwünsche fließen somit nur dann in die Open Source-Softwareentwicklung ein, wenn die Nutzer selbst zu Entwicklern werden, was dem ursprünglichen genossenschaftlichen Gedanken einer Open Source-Community entspricht. Dies bedeutet aber, arbeitsteilige Prozesse zurückzudrängen, Spezialisierungsgewinne aufzugeben und so die wichtigste Triebkraft für Produktivitätsfortschritt und wirtschaftliches Wachstum zu schwächen.

Das Beispiel des Webservers Apache zeigt, dass Open Source-Software zur Zeit vor allem dann sehr erfolgreich sein kann, wenn die Nutzer sehr technikaffin sind. In diesem Fall profitieren die Anwender von der Anpassungsfähigkeit der Software. Da sie selbst über ein entsprechendes technisches Know-how verfügen, sind Dokumentationen und eine einfache Handhabbarkeit nicht die entscheidenden Kriterien für den Einsatz der Software.[122]

Ein stärkeres Interesse bei der Berücksichtigung von Kundenwünschen haben Unternehmen, die an kommerziellen Geschäftsmodellen rund um Open Source beteiligt sind. Allerdings ist auch hierbei zu berücksichtigen, dass die entwickelte Software auch bei kommerziellen Geschäftsmodellen wieder frei zugänglich sein muss. Dies bedeutet aber, dass kein Unternehmen durch die Entwicklung einer besonders nutzerfreundlichen Software einen spezifischen Wettbewerbsvorteil erringen kann. Die auf Open Source aufbauenden Zusatzdienste sind dann erfolgreicher, wenn die zusätzliche Leistung exklusiv durch das anbietende Unternehmen zu vermarkten ist. Das Geschäftsmodell von Distributoren wie z. B. Suse oder RedHat kann dahingehend ausgelegt werden, dass sie versuchen, durch ihre Distributionen der Open Source-Software den Kundennutzen hinzuzufügen, den die reine Open Source-Entwicklung nicht hervorbringen kann. Allerdings sind hierbei die Möglichkeiten für die exklusive Vermarktung des so entstandenen Produktes ebenfalls sehr begrenzt, solange an restriktiven Lizenzmodellen wie der GPL festgehalten wird (vgl. hierzu auch 2.3.5.3). Die jüngsten Entwicklungen bestätigen diese Einschätzung. So hat RedHat den Support für die Consumer-Distribution „Red Hat Linux 9" eingestellt und konzentriert sich zukünftig nur noch auf die Enterprise-Variante, dessen Lizenzmodell sich stärker an kommerziellen Vorbildern orientiert. Auch die Übernah-

[121] Vgl. HANG UND HOHENSOHN (2003), S. 44 und EVANS UND REDDY (2002), S. 30.

[122] Vgl. LERNER UND TIROLE (2000), S. 8-9, SCHMIDT UND SCHNITZER (2003), S. 14.

me von Suse durch den Netzwerkspezialisten Novell deutet darauf hin, dass sich deren Geschäftstätigkeit anstelle des bislang defizitären Engagements im Consumer-Bereich künftig stärker auf Unternehmenskunden ausrichten wird.

2.3.5.2 Mangelhafte Ressourcenallokation

Durch den fehlenden Preismechanismus liefert das Open Source-Modell keine Informationen darüber, welchen Wert eine Software für den Nutzer hat. Der Vorteil eines Preissystems ist es, dass unterschiedliche Waren und Dienste mit einem einheitlichen Maßstab bewertet und damit vergleichbar gemacht werden können.[123] Im Open Source-Modell ist ein Dringlichkeitsvergleich für alternative Bedarfe nicht möglich. Dieses Bewertungsdefizit überträgt sich – wie im Folgenden gezeigt wird – unmittelbar auf die vorgelagerten Faktormärkte, auf denen im Open Source-Modell nicht nur keine Einkommen, sondern wiederum keine Knappheitssignale hervorgebracht werden können.

Wird die Open Source-Entwicklung als Freizeitbeschäftigung angesehen, ist die individuelle Zeiteinteilung jedem Entwickler freigestellt und unterliegt auch keinen ökonomischen Bewertungskriterien außer denen des Entwicklers selbst. Wird jedoch die Open Source-Entwicklungsmethode als grundsätzliche Alternative zum kommerziellen Modell angesehen, muss sie sich hinsichtlich der Ressourcenlenkung mit dieser vergleichen lassen. Nur so kann beurteilt werden, ob sie geeignet ist, die gewünschten Produkte mit dem geringst möglichen Aufwand hervorzubringen. Die bei der Open Source-Entwicklung entstehenden Knappheiten beziehen sich nicht auf den Zuteilungsmechanismus für die erstellten Produkte. Aufgrund der Nichtrivalität im Konsum kann es hierbei keine Knappheiten geben. Knappheiten entstehen beim Open Source-Modell durch die Lenkung der begrenzten Entwicklungskapazität in die richtige Verwendung. Soll das Open Source-Modell als generelle Alternative zur kommerziellen Welt angesehen werden, sind solche Knappheiten unvermeidbar.

Auch wenn Open Source-Software unentgeltlich abgegeben wird, ist die Entwicklung der Software aus volkswirtschaftlicher Sicht keineswegs kostenlos. Entwickler sehen sich Opportunitätskosten gegenüber, da sie die ihnen zur Verfügung stehende Zeit nur einmal verwenden können. Ohne Preissignal bleibt unklar, ob ihr Entwicklungseinsatz in einem Projekt A nicht in einem anderen Projekt B einen höheren Nutzen stiften könnte. Die traditionelle ökonomische Betrachtung von Software

als einem Gut ohne Rivalität unter den Nutzern gilt daher auch nur für bereits bestehende Software, also aus einer (ökonomisch weniger interessanten) Ex-post Perspektive. Wenn es aber darum geht, für die Produktion neuer Software knappe Ressourcen einzusetzen, herrscht sehr wohl Wettbewerb unter den potenziellen zukünftigen Nutzern, wenn sie vor die Wahl gestellt werden, auf die neue Software A zu verzichten, damit eine alternative Software B programmiert werden kann oder umgekehrt (Ex-ante Rivalität). Diesen Konflikt können Märkte einwandfrei über den Preismechanismus lösen, andere Koordinationsverfahren versagen hier, weil sie A und B mangels Bepreisung nicht bewerten und damit auch nicht vergleichen können.

Aus volkswirtschaftlicher Sicht führt das Fehlen von Informationen über die tatsächlichen Nachfragerwünsche daher zu einer falschen Lenkung von Ressourcen. Die eingesetzte Ressource ist hierbei die Entwicklerkapazität. In einem auf Preisen beruhenden Softwaremarkt wäre es für die Produzenten möglich, ihre Entwicklungsarbeit auf diejenigen Produkte zu konzentrieren, die am dringlichsten benötigt werden. Das Open Source-Modell kann den gewünschten Bedarf auf diesem Wege nicht erkennen. Damit ist es möglich, dass sehr viel Entwicklerkapazität in Produkte fließt, die nicht benötigt werden oder bei denen auch mit einem geringeren Entwicklereinsatz gute Ergebnisse erzielt werden könnten.[124] Die dargestellten Signalisierungsziele der Entwickler verstärken diese Tendenz zusätzlich. Dies macht es wahrscheinlich, dass Felder, die eine hohe Reputation versprechen, von vielen Entwicklern besetzt sind, während in anderen Bereichen kaum Entwicklungsarbeit geleistet wird. Aufgrund dieser nicht gegebenen Möglichkeit, Ressourcen in die richtige Verwendung zu lenken, ist das Open Source-Modell als grundsätzliche Produktionsmethode für Software ungeeignet.

Die Open Source-Entwicklungsmethode muss daher unter volkswirtschaftlichen Gesichtspunkten als weder effizient noch effektiv eingeordnet werden.[125] Sie ist weniger effektiv als das kommerzielle Modell, weil es keinen Mechanismus gibt der überprüft, ob die eingesetzte Entwicklerkapazität tatsächlich in eine sinnvolle Ver-

[123] Auch wenn vielfach behauptet wird, dass man „Äpfel nicht mit Birnen vergleichen kann": Der Preismechanismus ermöglicht genau einen solchen Vergleich.

[124] Zwar können solche Produkte auch auf dem kommerziellen Softwaremarkt entstehen, dort werden schlechte oder falsche Produkte jedoch vom Markt als Verlustbringer sanktioniert.

[125] Vgl. Effizienz ist das Verhältnis von Ergebnis zu Einsatz („Dinge richtig tun") Effektivität ist das Verhältnis von angestrebtem Nutzen zu tatsächlichem Nutzen („Die richtigen Dinge tun").

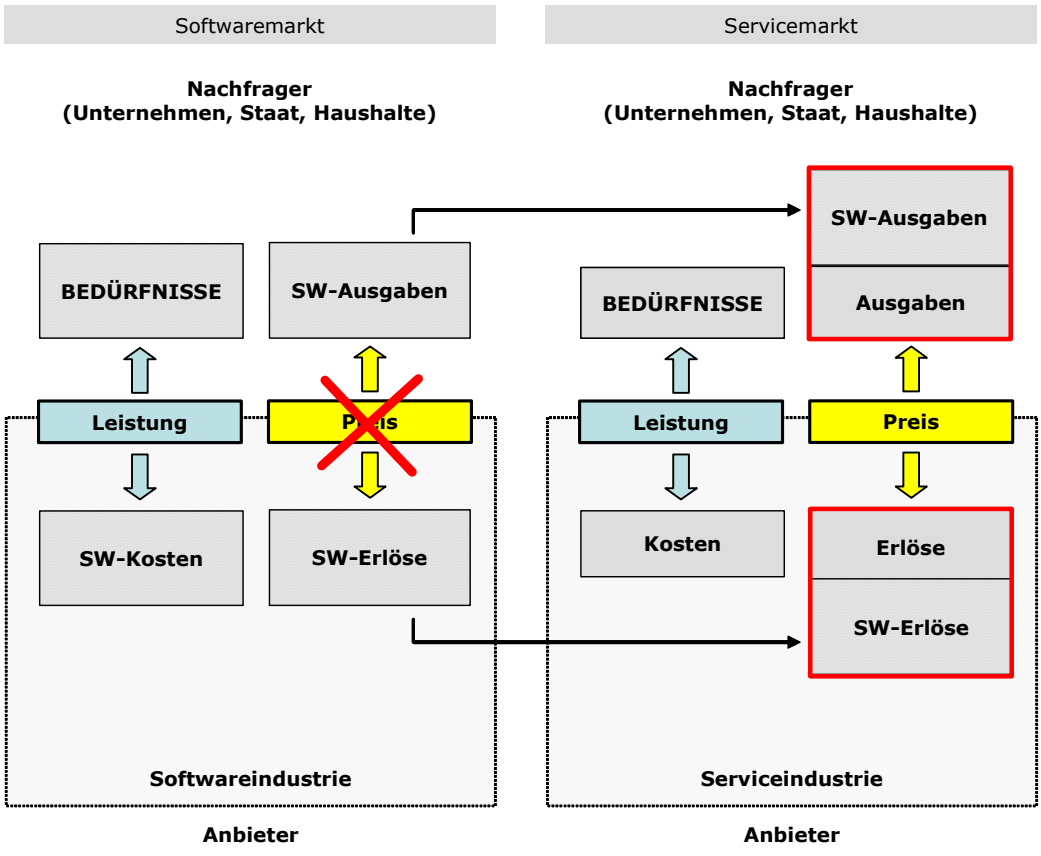
wendung fließt. Sie ist nicht effizient, weil der Einsatz im Verhältnis zum Ergebnis unangemessen hoch sein kann.

Die Ressourcenlenkung kann sich verbessern, wenn stärker kommerzielle Elemente in Open Source-Projekte eingebunden sind. Unternehmen werden genau in diejenigen Open Source-Projekte investieren, bei denen sie für ihr Geschäftsmodell den größten Zusatznutzen vermuten. Allerdings erfolgt hierbei die Ressourcenlenkung nur indirekt über den Nutzen, den das zu vermarktende Zusatzprodukt für das erstellende Unternehmen hat. Die Beurteilung derartiger Komplementärstrategien in Bezug auf die volkswirtschaftliche Koordinationseffizienz ist Gegenstand des folgenden Abschnitts.

2.3.5.3 Tragfähigkeit von Open Source-Komplementärstrategien

Zur Finanzierung der im marktfreien Open Source-Kern (vgl. Abbildung 7) anfallenden Entwicklungskosten wird zuweilen auf Open Source-Geschäftsmodelle verwiesen, die auf Open Source-Kernprodukte aufsetzen (Komplementärstrategie).

Abbildung 11: Komplementärstrategie mit Quersubventionierung



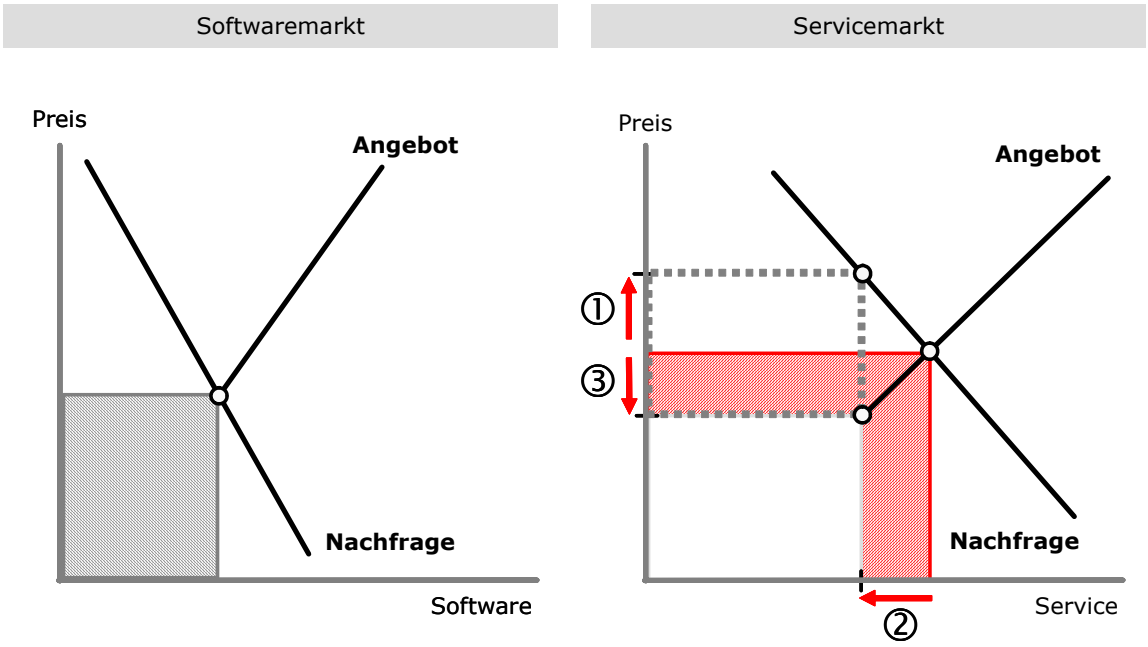
.

Dies läuft auf eine Quersubventionierung der Open Source-Softwareproduktion aus dem Erlösaufkommen aus nach- oder nebengelagerten Wertschöpfungsstufen hinaus. Zur volkswirtschaftlichen Bewertung dieser Strategie sind je nach Bestreitbarkeit der Komplementärmärkte zwei Fälle zu unterscheiden:

- Fall 1: Quersubventionierung ist möglich (keine Bestreitbarkeit)
- Fall 2: Quersubventionierung gelingt nicht (Bestreitbarkeit)

Zunächst soll Fall 1 näher betrachtet werden. Abbildung 11 zeigt die grundsätzliche Konstruktion dieses Finanzierungsmodells am Beispiel des Softwaremarktes (Open Source-Kern) und des darauf aufbauenden Servicemarktes als kommerziellem Komplementärmarkt.

Abbildung 12: Anpassungsreaktionen auf dem Komplementärmarkt



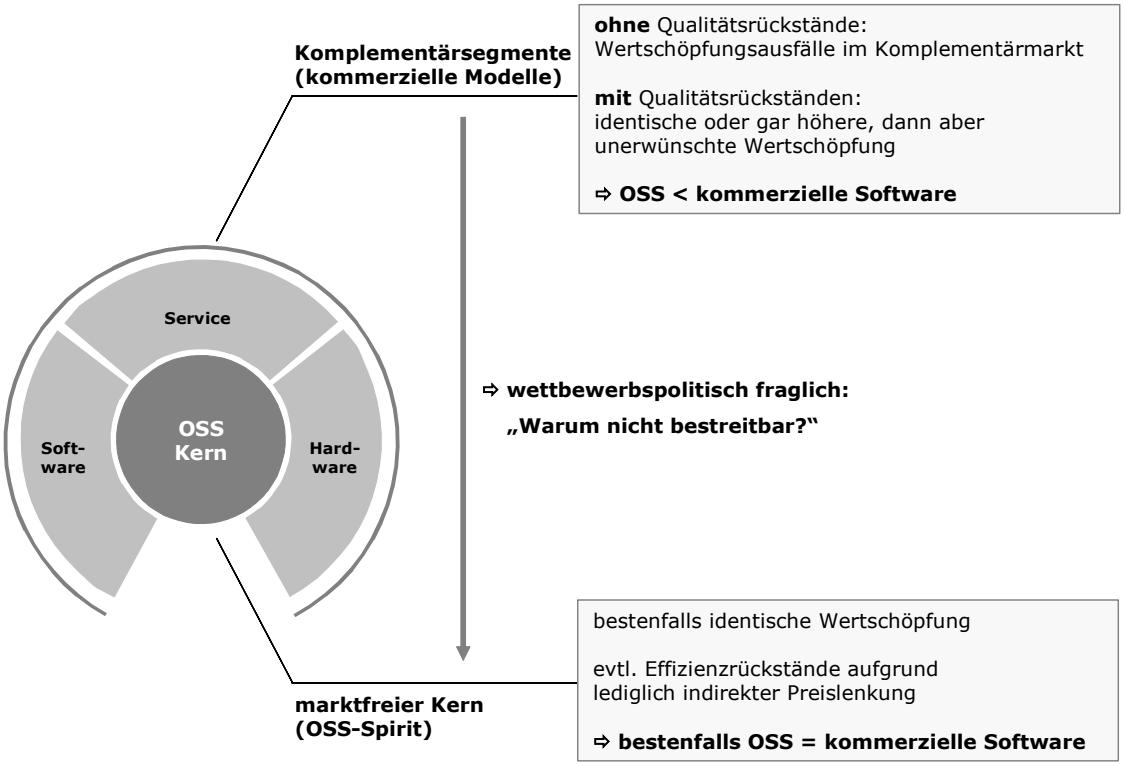
.

Da auf dem Softwaremarkt keine Bepreisung erfolgt, wird versucht, die gleichwohl anfallenden Kosten durch den Verkauf von Serviceleistungen zu Überkostenpreisen zu decken. Ausgehend von einem ursprünglichen Marktgleichgewicht (bei kommerzieller Organisation beider Märkte) bleibt die sich so ergebende Preiserhöhung auf dem Servicemarkt nicht ohne Anpassungsreaktionen, da die dortigen Nachfrager nicht bereit sind, die bisherige Gleichgewichtsmenge zu einem Preis zu kaufen, der deutlich über dem bisherigen Gleichgewichtspreis liegt. Die marktlichen Preis- und

Mengenreaktionen infolge der Entpreisung der Software und ihrer Quersubventionierung über einen Komplementärmarkt zeigt Abbildung 12.

Die schraffierte Fläche im linken Diagramm entspricht der durch die Entpreisung wegfallenden Wertschöpfung auf dem Softwaremarkt. Soll diese durch eine entsprechende Preiserhöhung auf dem Servicemarkt aufgefangen werden (gestricheltes Rechteck im rechten Diagramm), so führt dies zu einem Bruttopreisanstieg auf dem Komplementärmarkt (Effekt ①), wodurch sich die nachgefragte/absetzbare Menge reduziert (Effekt ②). Dies impliziert zugleich einen niedrigeren Nettopreis für Serviceleistungen (Effekt ③) und damit einen Wertschöpfungsrückgang auf dem Komplementärmarkt, der der schraffierten Fläche im rechten Diagramm entspricht. Insgesamt erhält man damit Wohlfahrtsverluste wie sie auch aus der Steuerwirkungsanalyse bekannt sind. Hier wie dort beruht der Wohlfahrtsverlust (dead weight loss) darauf, dass die relevanten Preise für Nachfrager und Anbieter auseinander fallen. Die Quersubventionierung wirkt wie ein Keil, der Kaufpreis und Absatzpreis auseinander treibt und damit über Mengenreaktionen die gesamtwirtschaftliche Wohlfahrt reduziert.

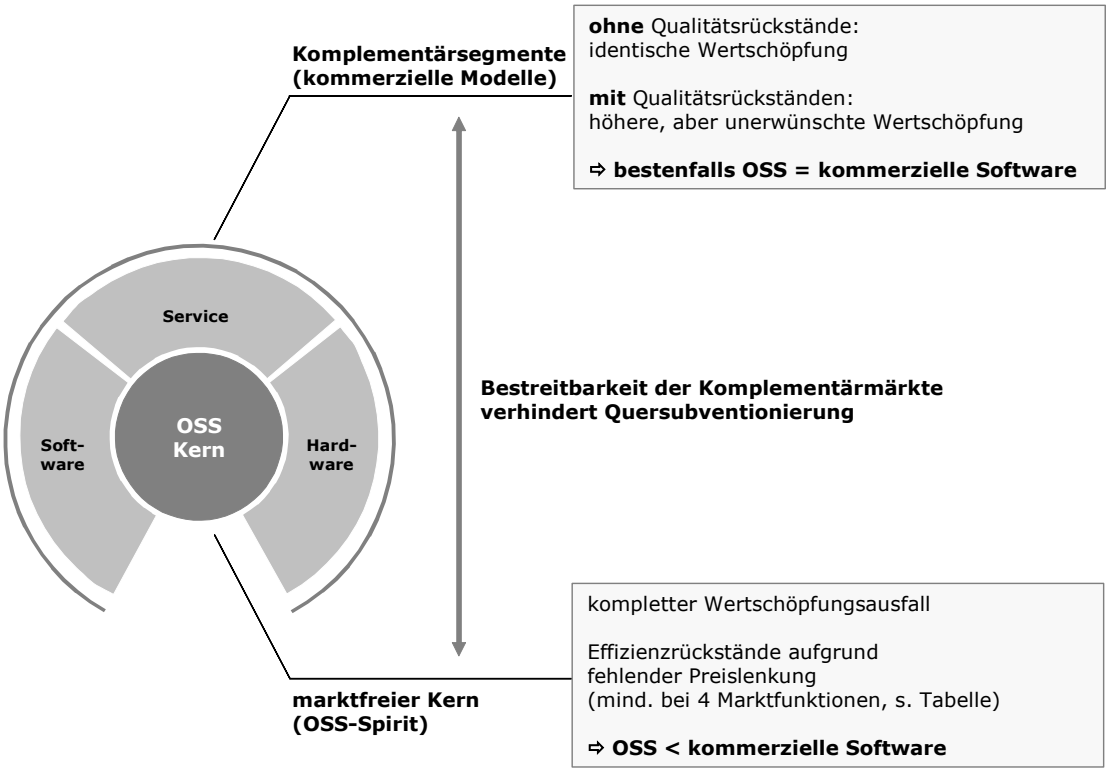
Abbildung 13: Bewertung der Komplementärstrategie mit Quersubventionierung



.

Die bisherigen Überlegungen gehen davon aus, dass ein Open Source-Softwareprodukt gleiche Serviceleistungen impliziert wie kommerzielle Software, beide Produktionsverfahren also zur selben Softwarequalität führen. Der Nettowertschöpfungsverlust den die Komplementärstrategie im gesamten IT-Sektor verursacht, könnte nur verhindert werden, wenn die Servicenachfrage für Open Source-Software größer wäre als für kommerzielle Software. Da sich aber Servicenachfrage und Produktqualität reziprok zueinander verhalten, impliziert dies, dass dem Wertschöpfungsverlust bei der Open Source-Softwareproduktion nur durch verschlechterte Softwarequalität (höhere Schulungsaufwendungen, mehr Support etc.) begegnet werden könnte. Diese Form der künstlichen Wertschöpfung wäre aber als volkswirtschaftlich unerwünscht einzustufen. Besser wäre es in diesem Fall, die Wertschöpfung in der Softwareentwicklung entstehen zu lassen, um die Volkswirtschaft mit qualitativ besserer Software zu versorgen, andernfalls kommt es – wie gezeigt – zu einer mangelhaften Durchsetzung der Konsumentensouveränität.

Abbildung 14: Bewertung der Komplementärstrategie ohne Quersubventionierung



.

Abschließend zu Fall 1 und überleitend zu Fall 2 stellt sich die Frage, warum eine Quersubventionierung überhaupt gelingen kann, da die Komplementärmärkte

grundsätzlich bestreitbar sind (falls nicht, wäre dies wettbewerbspolitisch problematisch). Da Open Source-Software jedermann auch ohne Finanzierungsbeitrag zugänglich sind, könnten z. B. Serviceanbieter am Markt auftreten, die ihre Leistungen zu Kostenpreisen anbieten und damit diejenigen Anbieter verdrängen, die Finanzierungsbeiträge für Open Source-Software über ihre Absatzpreise hereinholen müssen.

Bei Bestreitbarkeit der Komplementärmärkte (Fall 2) wird der Quersubventionierungsstrategie der Boden entzogen. Dies führt zu einem Wertschöpfungsausfall im IT-Sektor in Höhe der im linken Diagramm von Abbildung 12 schraffierten Fläche. Da nunmehr auch keine indirekte Verbindung zu kommerziellen Geschäftsmodellen mehr besteht, kommen die in Tabelle 4 genannten Koordinationsdefizite in der Open Source-Softwareproduktion voll zum Tragen. Soll der Wertschöpfungsausfall bei der Softwareproduktion durch höhere „Wertschöpfung" im Komplementärsektor kompensiert werden, so impliziert dies auch in diesem Fall schlechtere Produktqualitäten, wodurch dann wiederum das Ziel der Konsumentensouveränität verfehlt würde.

Verfolgt man die verschiedenen Zweige der hier vorgenommenen Fallunterscheidung (mit/ohne Quersubventionierung, mit/ohne Qualitätsrückstand), so stellt man fest, dass sich in der Summe im Open Source-Modell gegenüber der kommerziellen Softwareproduktion jeweils deutlich inferiore Ergebnisse einstellen. Dieses Gesamtergebnis ist letztlich auf die fehlende Bepreisung von Open Source-Kernprodukten zurückzuführen.

2.3.5.4 Geringere Innovationsfähigkeit

Innovation heißt: Neues Wissen wird verwendet, um Produkte oder Dienste anzubieten, die vom Kunden gegenüber den bisherigen Lösungen präferiert werden. Damit ist eine Innovation die Kommerzialisierung eines Produktes, das zuvor nicht kommerzialisiert war.[126] Innovation ist dabei auch im Sinne von Adaption zu verstehen, d.h. die Endverbraucher nehmen ein neues Produkt bzw. Verfahren an. In Abgrenzung dazu ist eine Erfindung die *Erschaffung* eines neuen Produktes oder Verfahrens. Um Innovation zu messen, muss man daher ihre Akzeptanz bzw. den Einfluss auf den privaten Verbraucher, Organisationen und die Gesellschaft im All-

[126] Vgl. hierzu IANSITI UND LERNER (2002), S. 2.

gemeinen analysieren. Daher sind Informationen über die Verbreitung eines Produktes zugleich Aussagen über die Innovationsstärke.[127]

In einer Volkswirtschaft sind Innovationen von großer Bedeutung. Üblicherweise werden Innovationen in Produktinnovationen und Verfahrens- bzw. Prozessinnovationen unterteilt. Bei einer Produktinnovation führt ein neues Verfahren zur Fertigung neuer Produkte bzw. Produktqualitäten. Verfahrensinnovationen ermöglichen demgegenüber die Einsparung von Produktionsfaktoren und damit eine Verringerung der Produktionskosten.[128]

Auf statischen Märkten[129] ist das Maß für die gesamtwirtschaftliche Wohlfahrt die Summe von Konsumenten- und Produzentenrente. Diese Summe wird auch als sozialer Überschuss bezeichnet, der dann maximal ist, wenn ein Gut zu Grenzkosten bepreist wird. Die Bepreisung von Open Source-Software zu Grenzkosten ist aus statischer Sicht effizient. Beim Software-Markt handelt es sich jedoch um einen dynamischen Markt, der durch rapide technologische Veränderungen gekennzeichnet ist. Eine Bepreisung zu Grenzkosten bietet in diesem Fall nicht genügend Anreize für Unternehmen, in die Softwareentwicklung zu investieren.[130]

Die bestehenden Anreize für professionelle Softwareentwickler, Fehler zu beheben oder die Software an bestimmte Bedürfnisse anzupassen, führen dazu, dass der private Nutzen der jeweiligen Entwickler steigt. Die Anreize zu Innovationen bei kommerzieller Softwareproduktion führen hingegen dazu, dass der gesamtwirtschaftliche Nutzen steigt.

Für ein Unternehmen stellen Innovationen immer einen kritischen Erfolgsfaktor dar, denn nur wenn sie erfolgreichere Innovationen hervorbringen als ihre Konkurrenten, werden sie sich auf dem Markt durchsetzen und Erträge erwirtschaften, die die Kosten decken. Firmen werden dann in die Entwicklung von Software investieren, wenn sie die geistigen Eigentumsrechte an der entstanden Software schützen können und wenn sie als Kompensation für mögliche Verluste im Falle eines Scheiterns mehr als die durchschnittlichen Kosten für das Produkt erzielen können.[131]

---

[127] Vgl. IANSITI UND LERNER (2002), S. 3. Siehe zum Zusammenhang zwischen Lizenzierung und Innovationen die Untersuchung von LERNER UND TIROLE (2002).

[128] Vgl. FRITSCH, WEIN UND EVERS (2003), S. 75.

[129] Auf statischen Märkten existiert eine gegebene Technologie und es kommt nicht zu Innovationen.

[130] Vgl. SCHMIDT UND SCHNITZER (2003), S. 6ff.

[131] Vgl. SCHMIDT UND SCHNITZER (2003), S. 8.

Ein Unternehmer hat daher nur dann einen Anreiz zur Durchführung von Innovationen, wenn er dadurch Vorsprungs– bzw. Pioniergewinne erzielen kann, die ihn für seine Aufwendungen, insbesondere für das mit der Innovationstätigkeit verbundene Risiko, entlohnen. Der Pioniergewinn ist der Ertrag, der daraus resultiert, dass kein anderes Unternehmen über die Innovation verfügt. Je länger die Anpassungsreaktionen seiner Konkurrenten, desto höher ist dieser Pioniergewinn des Innovators und damit auch der Anreiz zur Innovation. Sind seine Konkurrenten jedoch in der Lage, die Innovation ohne jede Zeitverzögerung zu imitieren bzw. muss er sie im Fall von quelloffener Software sofort zur Verfügung stellen, kann er keine Pioniergewinne realisieren. Eine sehr hohe Reaktionsgeschwindigkeit der Konkurrenten bei der Einführung neuer Produkte bzw. Produktionsverfahren würde gesamtwirtschaftlich dazu führen, dass die Innovationsanreize gemindert werden oder gar gänzlich zum Erliegen kommen.[132]

Die Aussicht auf zukünftige Gewinne ist daher der stärkste Anreiz zur Durchführung von Innovationen. Diese Anreize bestehen im Open Source-Modell nicht, denn durch die Offenlegung des Quellcodes hat jeder Anwender die Möglichkeit, die Software gemäß seinen Bedürfnissen zu verändern, außerdem kann sie beliebig oft kopiert und vertrieben werden. Unternehmen, deren Geschäft auf dem Open Source-Modell aufbaut, haben daher einen deutlich geringeren Anreiz, in F&E-Aktivitäten zu investieren und es ist fraglich, ob von ihnen die gleiche Innovationskraft ausgehen kann wie von einem Anbieter kommerzieller Software. Die für ein Unternehmen gegebenen Anreize in die Weiterentwicklung der Software zu investieren bestehen allenfalls indirekt, denn wie gezeigt können Unternehmen in der Open Source-Welt nicht mit der Software selbst, sondern nur mit zusätzlichen Diensten oder Produkten Erträge erwirtschaften.

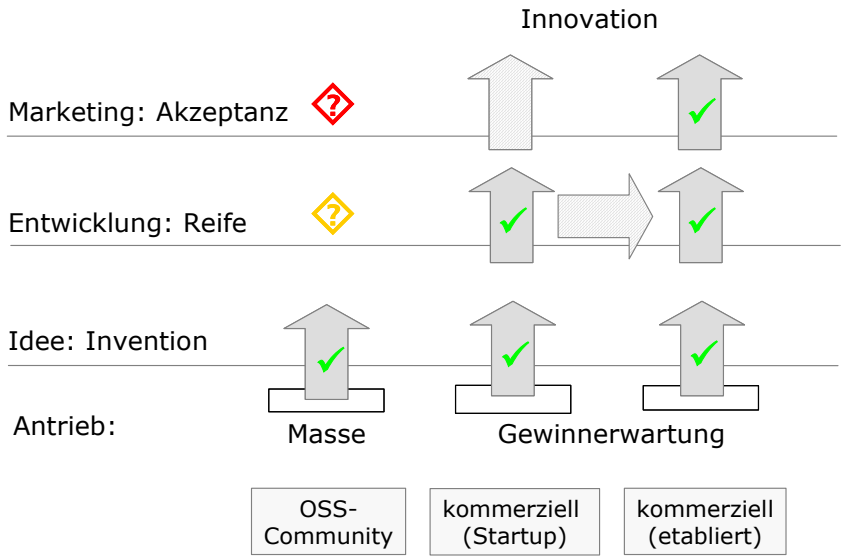
Zieht man beispielsweise zwei ansonsten identische Firmen als Beispiel heran, von denen die eine in die Open Source-Softwareentwicklung investiert und die andere nicht, so muss die erste Firma jede Softwareverbesserung die sie entwickelt (insbesondere unter der GPL), direkt der zweiten Firma zugänglich machen. Damit schafft der Open Source-Entwicklungsprozess zwar gleiche Geschäftsmöglichkeiten für beide Firmen, die erste Firma wird aber durch ihre Entwicklungsarbeit zwangsläufig höhere Kosten haben als die zweite Firma. Die erste Firma kann nur dann Erfolg haben, wenn die Open Source-Entwicklung ihr einen Vertrauensvorschuss der Kun-

[132] Vgl. FRITSCH, WEIN UND EVERS (2003), S. 77.

den verschafft oder sie kürzere Entwicklungszeiten für kommerzielle Software oder Dienste hat, die auf der Open Source-Software aufbauen. [133]

So liegt ein grundlegender Vorteil der kommerziellen Software-Entwicklung darin, dass sie den Entwicklern die Möglichkeit bietet, die getätigten Investitionen wieder zurück zu erhalten und den durch ihre Anstrengungen entstandenen zusätzlichen Nutzen für Konsumenten zumindest teilweise in eigene Verdienstmöglichkeiten zu lenken. *„Like in all other industries, the profit motive provides a very powerful incentive to innovate that is not present in the open source world"*[134].

Abbildung 15: Von der Idee zur Marktreife



Die Abbildung 15 vergleicht in einer vereinfachten Darstellung verschiedene Wege, die zu einer Innovation führen können. Damit eine Innovation entsteht, ist zunächst eine Erfindung oder Idee notwendig. Im Open Source-Modell entsteht diese aus der großen Masse der Entwickler. Dabei orientieren sich die Entwickler wie gezeigt an ihren eigenen Interessen oder Problemstellungen. Auf dieser Stufe ist das Open Source-Modell durch seine freie Form der Zusammenarbeit und die große Anzahl an Entwicklern in der Lage, eine Vielzahl an Ideen zu produzieren („Brainstorming-Effekt"). Diese werden jedoch keiner Auswahl hinsichtlich der Verwertbarkeit unterzogen. Demgegenüber orientieren sich kommerzielle Softwarehersteller auf dieser

[133] Vgl. EVANS UND REDDY (2002), S. 30.

[134] Siehe hierzu SCHMIDT UND SCHNITZER, S. 13.

Stufe an den Kundeninteressen, denn sie müssen zumindest die Erwartung haben, dass es genügend Abnehmer für ihr Produkt gibt, damit sich ihre Investitionen in die Entwicklung der Software lohnen. Das kommerzielle Entwicklungsmodell wird dabei unterschieden nach einem Startup- oder etablierten Unternehmen. Grundsätzlich sind auf dieser Stufe die Ausgangsvoraussetzungen für beide Unternehmenstypen gleich. Zahlreiche Softwareentwicklungen sind in der Vergangenheit als kleine Startups gestartet und haben mit ihrem Produkt beachtliche Marktanteile erreicht.

Inwieweit der Reifeprozess bei Open Source-Software vorangetrieben wird, hängt davon ab, wie viele Entwickler sich für die Weiterentwicklung eines Projekt zusammenfinden. Für das kommerzielle Startup-Unternehmen besteht ab dieser Stufe auch die Möglichkeit, von einem etablierten Unternehmen übernommen zu werden. Dabei bleibt die Anreizsituation grundsätzlich unverändert. Das etablierte Unternehmen wird nur dann ein Interesse an einem Aufkauf haben, wenn das Preis-Leistungs-Verhältnis aus Sicht seiner Nachfrager als besonders gut eingeschätzt wird.

Die letzte Stufe ist schließlich die Marktdurchdringung und weite Verbreitung des Produktes. Die Relevanz einer Innovation liegt nicht in der reinen Erfindung eines neuen Produktes oder Produktionsverfahrens. Eine Innovation ist nur dann von volkswirtschaftlicher Bedeutung, wenn sie weite Verbreitung findet und den Anwendern einen Nutzengewinn in Form von Produktivitätssteigerung oder Kostensenkung verschafft. Orientiert sich die Software-Entwicklung aber nicht an den Präferenzen der Anwender, wird sie auch keine Innovationen hervorbringen, die weite Verbreitung finden und damit wohlfahrtssteigernd in einer Volkswirtschaft wirken.

Hinsichtlich der Innovationsstärke muss sich Open Source-Software mit dem kommerziellen Modell messen lassen. Dabei lagen die Stärken von Open Source-Software bisher vorwiegend in der Kopie bereits bestehender Software. Schon der Start von Open Source-Software hatte die Kopie einer Software, nämlich des Unix-Betriebssystems, zum Ziel. *„Clearly, much innovation in commercial software has occurred over those 25 years. Just as clearly, much (but certainly not all) of the focus of GPL software over the past two decades has been on creating „free" versions of proprietary software*".[135]

[135] Vgl. EVANS UND REDDY (2002), S. 49.

2.3.5.5 Schwierigere Herausbildung von Standards

Wie im ersten Teil dargestellt kommen Standards[136] auf dem Softwaremarkt eine besondere Bedeutung zu. Sie ermöglichen es, Netzwerkeffekte auf der Angebots- und Nachfrageseite auszuschöpfen. Hiervon profitieren die Anbieter, die Software für einen Standard entwickeln, indem sie ihre Produkte nicht an verschiedene Plattformen anpassen müssen. Nachfrager profitieren von einem Standard, der ihnen z. B. den Dateiaustausch erleichtert und sie sich an verschiedenen Computer-Arbeitsplätzen nicht in unterschiedliche Plattformen einarbeiten müssen. Eine Standardisierung führt auch dazu, dass zwischen den Programmen verschiedener Anbieter definierte Schnittstellen festgelegt werden. Damit können programmübergreifende Funktionalitäten wie Copy und Paste gewährleistet werden. Im Open Source-Modell kann eine Herausbildung von Standards schwieriger sein als in der kommerziellen Welt, da die ökonomischen Anreize fehlen, einen Standard durchzusetzen und zusammenzuhalten. Das Problem der Netzzersplitterung und damit der Nicht-Durchsetzung eines Standards kann auf der Entwicklungs- und der Distributionsebene bzw. Herstellerebene auftreten.

Tritt dies bereits auf der Entwicklungsebene auf, spricht man von „Forking". Unterschiedliche Einschätzungen der Entwickler über die Richtung eines Projektes können möglicherweise zur Unterteilung in verschiedene Projekte führen, die unabhängig voneinander weiterentwickelt werden und nicht miteinander kompatibel sind.[137] Das Betriebssystem Unix ist das bekannteste Beispiel für das Entstehen vieler verschiedener und teilweise nicht kompatibler Versionen.[138] Dies wurde noch dadurch verstärkt, dass kommerzielle Anbieter jeweils eigene Unixversionen in Umlauf brachten, um sich vom jeweiligen Mitbewerber differenzieren zu können.

Zwar kann man „Forking" auch als das Konkurrieren verschiedener Entwicklungsrichtungen um die beste Lösung verstehen. Dabei steht jedoch wieder der Anspruch im Vordergrund, die technisch beste Lösung zu entwickeln. Für den Endnachfrager sind aber praktische Fragen der Kompatibilität oder der Austauschbarkeit von Dokumenten ebenfalls von großer Bedeutung. Da kommerzielle Anbieter an einer möglichst großen Verbreitung ihrer Software ein Interesse haben, werden sie ver-

[136] Unter „Standards" bzw. „Standardisierung" werden im Folgenden Maßnahmen zur Verhinderung von Forking-Effekten verstanden. Die Schwierigkeiten, die bei der Herausbildung offener IT-Standards wie TCP/IP auftreten können, werden hier nicht behandelt.

[137] Vgl. MENDYS-KAMPHORST (2002), S. 15 und 39.

[138] Wieder sehr plastisch spricht GRÖHN (1999), S. 11 hier von einer „Balkanisierung" von Unix.

suchen, auch eine entsprechend große Abdeckung durch ihre Produkte zu erreichen. Das Vorliegen einer gewissen Marktmacht ist zudem zur Etablierung eines Standards und Durchsetzung einer Plattform von Nutzen.

Auch auf der Ebene der Distributoren und Hard- bzw. Softwarehersteller, deren Geschäftsmodelle mit Open Source verknüpft sind, kann es zu einer Zersplitterung des Standards kommen. Im Bereich der Distributoren sind verschiedene Unternehmen darum bemüht, ihre eigenen Linux-Distributionen zu vertreiben. Es existiert mittlerweile eine Reihe von Linux-Versionen, die nicht zwingend miteinander kompatibel sein müssen.[139] So können die Distributoren verschiedene Ansätze wählen, um das Abspeichern von Dateien zu organisieren. Dies erschwert es den Entwicklern von Software-Anwendungen, eine Routine bei der Einrichtung von Anwendungen zu erstellen, da diese nicht auf verschiedenen Versionen läuft. Die Entwickler müssen somit für verschiedene Distributionen unterschiedliche Programmversionen anbieten bzw. diese müssen von den Distributoren angepasst werden.[140] Auch müssen Programme, die für eine Benutzeroberfläche lauffähig sind, nicht in der gleichen Form auf einer anderen Oberfläche laufen.[141]

Die große Bedeutung von Standards auf dem Softwaremarkt und die Notwendigkeit, diese auch in der Open Source-Welt zu etablieren, wird an der Bildung verschiedener Initiativen deutlich, die versuchen, einheitliche Regeln z. B. für die Dateiablage festzulegen.

2.4 Perspektiven für die Open Source-Entwicklung und wirtschaftspolitische Implikationen

2.4.1 Kommerzielle und Open Source-Software – Eine Gegenüberstellung

Ausgangspunkt der vorangehenden Analyse war das Fehlen eines Marktes als Koordinationsinstrument zwischen den Entwicklern bzw. Anbietern von Software und den Nutzern von Software. Da bei der Open Source-Entwicklung bewusst auf Marktprozesse verzichtet wird, bleiben wichtige volkswirtschaftliche Koordinationsaufgaben unerledigt, die der Markt bei kommerzieller Softwareproduktion über-

[139] Für einen Überblick über verschiedene Linux-Versionen siehe z. B. www.suse.de/de/index.html oder www.redhat.com/

[140] Vgl. MENDYS-KAMPHORST (2002), S. 48.

[141] EVANS UND REDDY (2002), S. 44 führen dieses Beispiel für die Oberflächen KDE und GNOME an.

nimmt. Der vorangehende Abschnitt hat die sich hieraus ergebenden Konsequenzen für verschiedene Marktfunktionen aufgezeigt. In der Tabelle 5 werden die wesentlichen Aussagen zu kommerzieller und Open Source-Software noch einmal gegenübergestellt.

Tabelle 5: Vergleich zwischen kommerzieller und Open Source-Software

	kommerzielle Software	**Open Source-Software**
Anbieter-Nachfrager-Koordination	Markt als Koordinationsinstrument	Autonome Softwareerstellung
Orientierung für Entwicklungen	Orientierung an den Kundenwünschen	Orientierung an den Entwicklerinteressen
Anreize für Innovationen	Gewinnstreben des Innovators Eigentum an Innovationen	Persönliche Interessen, die nicht deckungsgleich sein müssen mit Nutzerinteressen
Kompatibilität und Interoperabilität	Große Anreize zur Kompatibilität	Gefahr des Forking
Bugfixing	Leichte Installation, große Kompatibilität in unterschiedlichen Hardwarekonstellationen aber dadurch längerer Prozess	Schnelle Verfügbarkeit, aber benutzerfreundliche Installation und Kompatibilität evtl. schwierig
Anpassungsfähigkeit	Anpassungsfähigkeit innerhalb der definierten Möglichkeiten	Weitreichende Anpassungsmöglichkeiten für versierte Nutzer
Signaling	Abhängig von der Veröffentlichungspolitik des Herstellers	Großer Signaling Effekt durch Offenlegung des individuellen Entwicklerbeitrags

Die Unterschiede zwischen kommerzieller und Open Source-Software führen dazu, dass sich auch unterschiedliche Stärken der jeweiligen Softwaremodelle ergeben. Die **Stärken von kommerzieller Software** liegen bei:

1. **Schaffung von Innovation**

   Aufgrund der Zuordnung von Eigentumsrechten bestehen auf dem kommerziellen Softwaremarkt große Anreize zu Investitionen in Forschung und Entwicklung und zur Schaffung von Innovationen. [142] Demgegenüber stehen im Open Source-Modell alle Entwicklungsergebnisse allen anderen Beteiligten offen. Wenn es nicht um eine Vielzahl kleinerer Änderungen geht, sondern um grundsätzliche

[142] Vgl. SMITH (2002), S. 77.

Entwicklungslinien, bietet die kommerzielle Softwareentwicklung Organisationsstrukturen, die eher in der Lage sind, eine Marktreife zu realisieren.

2. Nutzer- bzw. Kundenorientierung

Die Erstellung von kommerzieller Software ist kundenorientiert, demgegenüber liegt bei Open Source-Software eine Entwicklerorientierung vor. Open Source-Entwickler orientieren sich vielfach an technischen und nicht kundenorientierten Maßstäben wie Benutzerfreundlichkeit und Bedienbarkeit. Während bei kommerzieller Software der Marktmechanismus dafür sorgt, dass Software den Kundenwünschen entsprechend entwickelt wird, gibt es bei Open Source-Software keinen Anreizmechanismus, der die Entwicklerinteressen auf die Kundenwünsche lenkt.

3. Benutzerfreundlichkeit: Große Usability und Easy to use

Kommerzielle Software muss sich bei Standardanwendungen für den Massenmarkt an einer möglichst breiten Kundenbasis orientieren. Um eine große Marktabdeckung zu erreichen, müssen auch nicht-technikaffine Nutzer die Produkte bedienen können. Hierzu gehört auch eine umfangreiche Dokumentation der Software, an der es bei Open Source-Software häufig mangelt. Über die Qualität eines kommerziellen Produktes und damit den Erfolg eines Unternehmens entscheiden die Verkaufszahlen und damit die Kundenzufriedenheit.[143]

4. Hohe Standardisierung, Kompatibilität und Updates

Kommerzielle Softwareanbieter haben ein Interesse an einer hohen Standardisierung ihrer Produkte. So wird eine Kompatibilität in verschiedenen Hardwareumgebungen erreicht. Die Entwickler von Soft- und Hardware können davon ausgehen, dass ihre Produkte mit definierten Plattformen immer in der gleichen Weise zusammenarbeiten. Hierzu gehören auch die Verfügbarkeit von Treibern und deren leichte Installation sowie das regelmäßige Update der Software und die Anpassung an neue Hardwarekonstellationen.

5. Keine Fragmentierung

Da kommerzielle Softwareanbieter die kompletten Verfügungsrechte über den Quellcode der von ihnen entwickelten Software besitzen, verfügen sie auch über die technischen Voraussetzungen, eine Fragmentierung der Software und das

[143] Siehe zur Benutzerfreundlichkeit HORST (2003), S. 36.

Entstehen inkompatibler Versionen zu verhindern. Zudem bestehen für sie Anreize, die Kompatibilität mit möglichst vielen bereits installierten Versionen zu erhalten und so die Netzwerkeffekte für Nutzer zu schützen und auszubauen.

6. **Breites Softwareangebot im Applikationsbereich**

   Für kommerzielle Softwareplattformen gibt es ein nahezu unendliches Angebot an Applikationen. Es ist gewährleistet, dass diese Applikationen in verschiedenen Hardwarekonstellationen für die definierten Plattformen immer gleich funktionsfähig sind.

Die **Stärken von Open Source-Software** ergeben sich aus den folgenden Punkten:

7. **Anpassungsfähigkeit für technikaffine Nutzer**

   Open Source-Software ermöglicht aufgrund des offenen Quellcodes eine unbegrenzte Anpassungsfähigkeit der Software. Von diesem Vorteil profitieren allerdings nur diejenigen Nutzer, die auch über das entsprechende Wissen verfügen, um solche Anpassungen vorzunehmen. Für den Massenmarkt spielt die Anpassungsfähigkeit einer Software kaum eine Rolle, da die Nutzer nicht über das hierfür erforderliche technische Wissen verfügen.

8. **Schnelle Wissensverbreitung**

   Aufgrund der fehlenden Eigentumsrechte und der freien Zugänglichkeit des Quellcodes kann im Open Source-Modell eine einfachere unbeschränkte Wissensverbreitung erfolgen. Das innerhalb des Open Source-Modells produzierte Wissen steht sofort allen anderen Entwicklern zur Verfügung. Allerdings ist darauf hinzuweisen, dass die fehlenden Eigentumsrechte auch zu einer geringeren Wissensproduktion führen.

9. **Signaling**

   Open Source-Software legt den Entwicklungsbeitrag jedes einzelnen Programmierers offen. Damit ist die Open Source-Programmierung besser für einen Entwickler geeignet, seine Entwicklerqualitäten öffentlich zu signalisieren. Allerdings erfolgt auch bei verschiedenen kommerziellen Softwareprodukten die Angabe der beteiligten Entwickler.

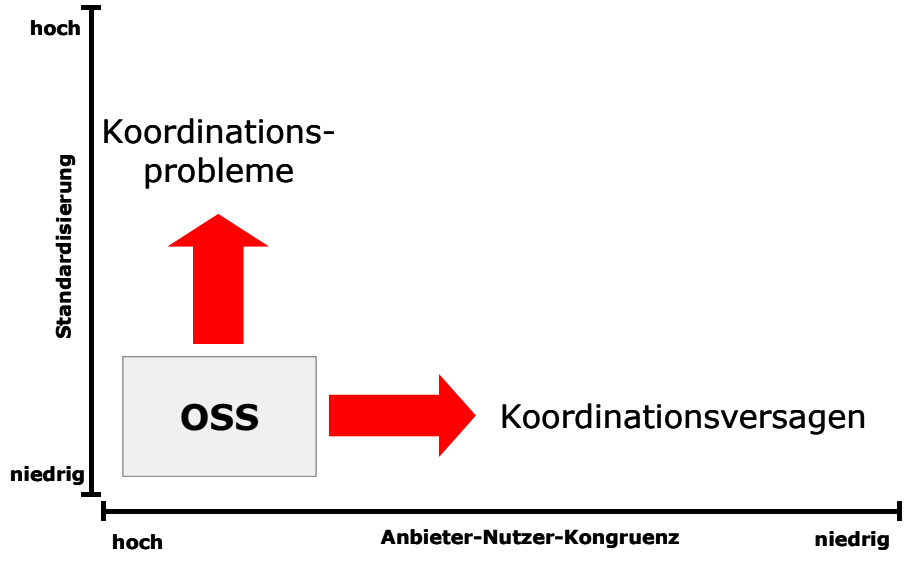
Die unterschiedlichen Stärken der beiden Softwaremodelle spiegeln sich im jeweiligen Produktangebot wieder. So lassen sich die Produktgruppen identifizieren, bei denen das jeweilige Softwaremodell im Vorteil ist. Im Folgenden wird die Bedeu-

tung der Merkmale Standardisierung und Entwicklerorientierung für das Softwareangebot genauer untersucht und in der Abbildung 16 abgetragen.

Je größer die **Standardisierung** und Kompatibilitätsanforderungen an eine Software sind, desto eher treten im Open Source-Modell Koordinationsprobleme auf. Für die Durchsetzung eines Standards und die Gewährleistung der Kompatibilität ist ein abgestimmtes Handeln notwendig, dass sich eher in den Strukturen und Anreizen des kommerziellen Modells erreichen lässt.

Bei der **Entwicklerorientierung** können Entwickler und Nutzer eine deckungsgleiche Gruppe sein oder unterschiedlichen Gruppen angehören. Sind die Nutzer auch gleichzeitig Entwickler, so sind Nutzerinteressen auch automatisch Entwicklerinteressen. Sofern Nutzer nicht Entwickler sind, muss die Orientierung der Entwickler an ihren Interessen nicht zu einem Produkt führen, dass auch den Nutzerinteressen entspricht. Damit steigen die Koordinationsprobleme bei der Softwareentwicklung, in Form der Abstimmung zwischen Nutzerinteressen und Produktangebot. Im kommerziellen Modell sorgt der Marktmechanismus für diese Interessenabstimmung. Je weiter die Nutzer und Entwicklergruppen auseinander fallen, desto gravierender ist das im Open Source-Softwarebereich auftretende Koordinationsversagen.

Abbildung 16: Einsatzbereiche der verschiedenen Softwaretypen



.

Open Source-Software kann damit für einzelne Bereiche ein adäquates Produktangebot hervorbringen (niedrige Standardisierungsnotwendigkeit, hohe Anbieter-Nutzer-Kongruenz). Sie war in der Vergangenheit insbesondere in den Bereichen

erfolgreich, in denen die oben beschriebenen Merkmale nur eine geringe Relevanz hatten. Open Source-Software hat bei den Produkten Vorteile, bei denen die Nutzer auch gleichzeitig über Entwicklerkenntnisse verfügen, bei denen Standards- und Kompatibilität eine geringere Rolle spielen und bei denen keine grundlegend neuen Innovationen erforderlich sind. Eine Massenmarkttauglichkeit des Open Source-Modells ist jedoch nicht gegeben. Insbesondere hinsichtlich der langfristigen Tragfähigkeit ist das kommerzielle Modell deutlich überlegen. Es ist besser als das Open Source-Modell in der Lage, Kundenwünsche zu identifizieren und Innovationen hervorzubringen.

Als limitierender Faktor kommt die Abhängigkeit des Open Source-Modells vom kommerziellen Softwaremarkt hinzu. Die Darstellung der Entwicklermotivation hat gezeigt, dass Signalisierungsanreize in Richtung des kommerziellen Softwaremarktes ein Motiv zur Partizipation sein können. Die in einem Open Source-Projekt aufgebaute Reputation stiftet nur dann einen Nutzen, wenn sie monetär verwertbar ist. Es muss daher ein kommerzieller Markt existieren, auf dem die Entwickler für ihre Leistung entlohnt werden. Auf diesem Markt müssen Unternehmen Erträge generieren, die es ihnen ermöglichen, eine große Anzahl an Entwicklern beschäftigen zu können. Besteht für die Open Source-Entwickler keine Möglichkeit, ihre Reputation auf einem kommerziellen Markt einsetzen zu können, fehlt ihnen auch ein wichtiger Anreiz, zur Entwicklung von Open Source-Projekten beizutragen. Damit kann die Bereitschaft an Open Source-Projekten teilzunehmen abnehmen, je kleiner der kommerzielle Markt wird.

Auch die bestehenden kommerziellen Geschäftsmodelle auf Basis von Open Source-Software sind kein Garant für den langfristigen Erfolg von Open Source-Software und eine mögliche Massenmarkttauglichkeit. Wie dargestellt hängt der Erfolg dieser Geschäftsmodelle von der Vermarktung von komplementären Produkten ab, wobei diese Strategie in allen hier untersuchten Verzweigungen insgesamt zu volkswirtschaftlich inferioren Lösungen führt. Die Anreize, in die eigentliche Open Source-Entwicklung zu investieren, sind auch bei kommerziellen Geschäftsmodellen begrenzt.

Das Open Source-Entwicklungs-Modell ist somit aus ordnungspolitischer Sicht nicht geeignet, an die Stelle des kommerziellen Modells zu treten. Für den weitaus größten Teil des Softwaremarktes ist das kommerzielle Modell besser geeignet, die gewünschten Produkte hervorzubringen. Dennoch gibt es eine Vielzahl an Initiati-

ven, die den Einsatz von Open Source-Software fördern. Ob eine solche Förderung aus volkswirtschaftlicher Sicht sinnvoll ist, untersuchen die nächsten Abschnitte.

2.4.2 Motive zur Förderung von Open Source-Software

In jüngster Zeit haben immer mehr Staaten damit begonnen, den Einsatz von Open Source-Software zu fördern.[144] Da der Staat selbst großer Softwarenachfrager ist, kann eine Förderung von Open Source-Software erfolgen, indem Beschaffungsentscheidungen bewusst zugunsten von Open Source-Software erfolgen. Auch wenn ein neutraler Kostenvergleich (TCO, Total Cost of Ownership) zu dem Ergebnis kommt, dass der Einsatz kommerzieller Software bei Einbeziehung aller relevanten Kosten insgesamt günstiger ist, erfolgt dennoch eine Entscheidung für Open Source-Software.

So basiert die Entscheidung der Stadt München für Linux nicht auf ökonomischen, sondern vielmehr auf „qualitativ-strategischen" Argumenten. Die Studie der Unternehmensberatung Unilog kommt zu der Empfehlung, dass „die Aktualisierung der heute eingesetzten Microsoft Produkte auf die nun aktuellen XP-Versionen die technisch einfachste und wirtschaftlich sinnvollste Handlungsalternative für die LHM darstellt".[145] So überwiegen die monetären Vorteile der Alternative Windows XP gegenüber der nächst teureren Ausstattungsalternative mit ca. 2,46 Mio. Euro beim Kapitalwert und mit ca. 1,76 Mio. Euro beim Gesamtwert. Der Kostenvorteil von Windows XP gegenüber einer reinen Linux-Lösung beläuft sich auf ca. 11,9 Mio. Euro beim Kapital- und ca. 11,6 Mio. Euro beim Gesamtwert.[146] So ergibt sich aus betriebswirtschaftlicher Sicht ein eindeutiger Vorteil für die kommerzielle Lösung. Aus „qualitativ-strategischer" Sicht wird aber die Open Source-Lösung als vorteilhafter bewertet. Sie soll eine höhere Grundsicherheit und eine geringere Herstellerabhängigkeit gewährleisten, da die Software-Produkte *„nicht von einem Hersteller, sondern von einer unabhängigen, aus vielen Entwicklern bestehenden Gruppe entwickelt werden*".

Auch die Initiative „Bundestux", die im Vorfeld der Entscheidung für eine neue IT-Infrastruktur des Bundestages den Einsatz von Linux förderte, sah nicht reine Kos-

[144] Derzeit liegen über 60 Regierungsinitiativen, -studien und -erklärungen in 25 Ländern vor, die den Einsatz von Open Source forcieren sollen. Siehe hierzu http://www.softwarechoice.org.
Einen Überblick verschiedener staatlicher Aktivitäten bietet auch HAHN (2002).

[145] Vgl. UNILOG INTEGRATA (2003), S. 29.

[146] Vgl. EBENDA, S. 19.

tenabwägungen als ausschlaggebendes Kriterium für die Beschaffungsentscheidung an. Die Initiative bezeichnete die "*Einführung eines freien Betriebssystems im Deutschen Bundestag aus ordnungs-, wettbewerbs- und standortpolitischen sowie demokratischen Gründen*" als ein "*notwendiges Signal für Deutschland*".[147]

Die Machbarkeitsstudie der Unternehmensberatung INFORA empfiehlt letztendlich den Einsatz von Linux im Bundestag nur im Bereich der E-Mail-Server und als Groupware-Lösung. Für alle anderen Serverdienste sowie Desktop-Anwendungen sei man mit Windows aufgrund der noch nicht ausreichenden Funktionen von Open Source-Software für die Endanwender besser bedient. Entgegen dieser Empfehlung entschied sich der Bundestag dennoch dafür, Linux auch auf den Datei- und Druckservern einzusetzen und den zentralen Verzeichnisdienst bereitstellen zu lassen, obwohl diese Lösung rund 80.000 Euro mehr als das von Infora favorisierte Konzept kostet. Die Entscheidung wurde damit begründet, dass man bei zukünftigen Anschaffungen nicht auf Microsoft-kompatible Produkte beschränkt sei und dieser Mehraufwand sich daher schnell auszahlen würde, „*man bringe sich mit ein bisschen mehr Geld in eine vorteilhafte strategische Position*".[148]

Die folgenden Abschnitte untersuchen, inwieweit die Förderung von Open Source-Software tatsächlich eine staatliche Aufgabe sein kann. Dazu wird nach möglichen ordnungs-, wettbewerbs-, oder standortpolitischen Argumenten unterschieden.

2.4.3 Die Förderung von Open Source-Software

2.4.3.1 Keine ordnungspolitische Aufgabe

Unter Ordnungspolitik werden alle Regeln und Handlungen verstanden, die es ermöglichen, die Wirtschaft nach den Prinzipien von Markt und Wettbewerb zu organisieren. Wesentlich ist dabei, den Wettbewerb zu sichern und den einzelnen ökonomischen Akteuren die notwendigen Freiräume für ihre wirtschaftliche Betätigung zu geben. Erforderlich ist hierfür eine sinnvolle Abgrenzung zwischen den Tätigkeiten des Staates und der privaten Wirtschaft.

In einer Marktwirtschaft kommt dem Staat die allgemeine rahmensetzende Funktion zu. Darüber hinaus soll der Staat nur dann durch Regulierungen oder eigene wirtschaftliche Tätigkeit in den Marktprozess eingreifen, wenn die Funktionsfähig-

[147] Vgl. www.bundestux.de

[148] Vgl. o. V. (2002). Siehe hierzu: http://www.heise.de/newsticker/data/odi-27.02.02-000/

keit der Märkte nicht gewährleistet ist. Die Wettbewerbspolitik ist Teil der Ordnungspolitik. Im Mittelpunkt der ordnungspolitischen Analyse in diesem Abschnitt steht die Untersuchung, ob auf dem Softwaremarkt besondere Funktionsdefizite vorliegen, die ein staatliches Eingreifen rechtfertigen könnten. Hiervon unterschieden werden im nächsten Abschnitt wettbewerbspolitische Gründe für ein Eingreifen des Staates. Dort wird untersucht, ob die auf dem Softwaremarkt bestehenden Marktpositionen ein Einschreiten notwendig machen und ob die Open Source-Förderung hierfür eine geeignete Maßnahme wäre.

Aus ökonomischer Sicht müssen zwei Kriterien erfüllt sein, um staatliche Eingriffe auf einem bestimmten Markt zu legitimieren:

1. Auf einem Markt liegt ein Marktversagen vor.
2. Der Staat kann eine effiziente und kostengünstige Lösung für dieses Versagen bereitstellen, und der Nutzen, der durch die Beseitigung des Versagens generiert wird, ist höher als die entstehenden Kosten.

Zu 1: Zunächst ist zu untersuchen, ob auf dem Softwaremarkt ein Marktversagen vorliegt. Theoretisch könnte ein Marktversagen in Form von Nicht-Exkludierbarkeit und Netzwerkeffekten vorliegen.

Wie die Kollektivgütertheorie gezeigt hat, ist die Exkludierbarkeit Voraussetzung für die Entstehung eines Marktes. Eine **Nicht-Exkludierbarkeit** wäre die Folge unzureichend definierter bzw. nicht hinreichend durchsetzbarer Eigentumsrechte. Liegt eine Nicht-Exkludierbarkeit vor, kann der Staat versuchen, Eigentumsrechte zu etablieren oder – falls dies nicht möglich ist – selbst als Produzent oder Beschaffungsagent auftreten. Wie die Entwicklungsgeschichte des Softwaremarktes gezeigt hat, ist der kommerzielle Softwaremarkt genau dadurch entstanden, dass ein (zumindest rechtlich möglicher) Ausschluss nicht zahlungswilliger Nutzer durch die Schaffung von Urheberrechten auch auf dem Softwaremarkt möglich wurde. Da aber gerade die freie Verfügbarkeit das Hauptkriterium von Open Source-Software darstellt, widerspricht ein solcher Eingriff der Open Source-Idee.

Im Hinblick auf die Netzwerkeffekte kann nach dem Vorliegen „starker" Netzwerkeffekte und „schwacher" Netzwerkeffekte unterschieden werden.[149] Liegen **starke Netzwerkeffekte** vor, so dominieren die Netzwerkeffekte die Entscheidung der Nutzer. Auch wenn eine Software überlegen ist, werden sich die Nutzer derjenigen

Software anschließen, von der sie erwarten, dass dies auch alle anderen tun. Damit kann sich theoretisch eine Entscheidung für eine unterlegene Technologie ergeben. Voraussetzung hierfür ist, dass genügend Nutzer erwarten, dass sich eine Mehrheit dieser neuen Technologie anschließt.

Allerdings gibt es auch Mechanismen, die darauf hindeuten, dass sich Nutzer trotz der dominanten Netzwerkeffekte für die überlegene Technologie entscheiden. So ist davon auszugehen, dass sich zum Zeitpunkt der Herausbildung eines Netzwerkes die Nutzer primär an der Qualität des Produktes orientiert haben. Die weiteren Nutzer sind dann genau dieser Entscheidung gefolgt. Es ist auch denkbar, dass die Entscheidung der Nutzer durch einen Erfahrungsaustausch und Testberichte indirekt koordiniert wird.

Insgesamt wird sich auf einem Markt mit starken Netzwerkeffekten ein dominantes Netz entwickeln. Sofern sich die Nutzer für das richtige Netz entschieden haben, wäre ein solches Ergebnis volkswirtschaftlich effizient. Wenn sich nach Entstehung eines Netzwerkes eine bessere Technologie herausbildet, so ist ein Wechsel zu dieser nur dann sinnvoll, wenn die Wechselkosten in Form der spezifischen Investitionen in Umstellung und Weiterbildung mit berücksichtigt werden. Nur wenn der Nutzen des neuen Netzwerks größer ist als der Nutzen der alten Technologie incl. der Wechselkosten, lohnt sich ein Netzwechsel.

Sofern davon ausgegangen wird, dass ein neues Netz aufgrund der Unterlegenheit der bisherigen Technologie insgesamt sinnvoll wäre, müsste allerdings bereits vor der Marktdurchsetzung der neuen Technologie deren Überlegenheit bekannt sein. Dieser Nachweis ist allerdings schwierig zu erbringen, denn weder staatliche noch private Institutionen können im Voraus entscheiden, welche Lösung die technisch überlegene ist.[150] Es kann bezweifelt werden, dass gerade staatliche Instanzen in der Lage sind, die richtige Technologieentscheidung für andere zu treffen.[151]

Selbst bei einer wie auch immer zustande gekommenen neutral-objektiven Bewertung zugunsten einer neuen Technologie müssen die beschriebenen Investitionen, die jeder Nutzer durch seine Ausbildung und sein Wissen über den Umgang mit die-

[149] Vgl. SCHMIDT UND SCHNITZER (2002), S. 15-19.

[150] Oder in der Originalquelle bildlicher gesprochen *„Staatliche wie private Institutionen sind im gleichen Maße Wanderer, die im Nebel der zukünftigen Entwicklung ihren Weg suchen müssen*." GRÖHN (1999), S. 54.

[151] "*…governments do not have good track records at picking technology winners and loosers*" EVANS UND REDDY (2002), S. 71.

ser Technik getätigt hat, mit berücksichtigt werden. Erst wenn der Nutzen der neuen Technologie so groß ist, dass er den Nutzen der alten Technologie plus der getätigten spezifischen Investitionen übersteigt, ist ein Wechsel ökonomisch sinnvoll.

Greift der Staat in einen Markt mit starken Netzwerkeffekten ein, so kann er eine Vorentscheidung zugunsten einer Technologie treffen und den Markt zu einem wesentlichen Anteil in Richtung dieser Software kippen lassen. Damit werden durch einen Eingriff in den Markt die Anreize für kommerzielle Unternehmen, in die Entwicklung von Software zu investieren, verringert. Es kommt hinzu, dass die Zusammenarbeit zwischen kommerzieller und Open Source-Software dann eingeschränkt ist, wenn die Open Source-Software unter der GPL lizenziert ist. Damit würde das Open Source-Netzwerk parallel neben einem kommerziellen Netzwerk entstehen. Die Förderung von Open Source-Software auf einem Markt mit starken Netzwerkeffekten würde damit ein zweites Netzwerk in einem Markt etablieren, in dem ein singuläres Netzwerk ökonomisch überlegen wäre.[152]

Liegen **schwache Netzwerkeffekte** vor, kann sich der Wettbewerb durch den Eingriff des Staates sogar noch reduzieren. Im Falle schwacher Netzwerkeffekte können zwei Netzwerke nebeneinander entstehen. Es gibt dann Konsumenten, die sich unabhängig von Netzwerkeffekten für die eine oder andere Plattform entscheiden. Daneben gibt es eine dritte Gruppe an Konsumenten, die keine Präferenzen für eine Plattform haben und um die somit Wettbewerb zwischen den Plattformen besteht.

Präferiert der Staat in diesem Umfeld Open Source-Software, so beeinflusst er genau die Nutzer, die keine grundsätzlichen Präferenzen für eine Technologie haben. Damit reduziert sich die Marktgröße der Konsumenten, um die Wettbewerb herrscht. Die Konsequenz hieraus ist, dass die Preise für kommerzielle Software steigen und Investitionsanreize im kommerziellen Markt beschädigt werden.[153]

Zu 2: Ergibt die Analyse Anzeichen für das Vorliegen eines Marktversagens, rechtfertigt dies jedoch nicht automatisch einen Eingriff des Staates, denn es besagt nicht, dass der Staat auch in der Lage ist, eine Allokationsverbesserung oder gar das Optimum herbeizuführen. Es gibt mehrere Gründe, die gegen ein staatliches Handeln sprechen. Es kann als sicher gelten, dass die staatlichen Entscheidungsträger nicht über die relevanten Informationen (Präferenzen der Haushalte, Kosten-

[152] Vgl. SCHMIDT UND SCHNITZER (2002), S. 26-27.

strukturen von Unternehmen, Entwicklungspotenzial neuer Technologien) verfügen, um eine Verbesserung des Allokationsergebnisses herbeizuführen. Hinzu kommt, dass staatliche Korrekturmaßnahmen oftmals eine Veränderung der Anreizstruktur der Privaten bedingen. Inwiefern es gelingt, diese Anreize so zu beeinflussen, dass daraus eine Verbesserung der Allokation resultiert, ist schwer abzuschätzen. Häufig impliziert eine Korrektur des Marktversagens, dass marktliche Allokationsentscheidungen durch staatliche Entscheidungsverfahren ersetzt werden. Die Erfahrung zeigt jedoch, dass staatliche Entscheidungsträger sich in vielen Fällen nicht ausschließlich an der gesamtwirtschaftlichen Effizienz, sondern – bedingt durch bürokratieinterne Anreizstrukturen – durchaus auch an ihren eigenen Interessen orientieren.[154]

Damit lassen sich keine Anzeichen finden, dass aufgrund eines Marktversagens ein Eingriff zugunsten von Open Source möglich wäre. Ein Eingriff des Staates könnte vielmehr dazu führen, dass der Wettbewerb reduziert wird, Netzwerkeffekte sich verringern oder der Markt zugunsten einer Technologie kippt, die nicht die überlegene Technologie sein muss.

2.4.3.2 Keine wettbewerbspolitische Aufgabe

Die Förderung von Open Source-Software wird teilweise auch als eine wettbewerbspolitische Maßnahme angesehen. Hierbei steht vor allem die Marktposition von Microsoft im Mittelpunkt der Argumentation. Durch die Förderung einer zweiten Betriebssystemsplattform soll die Marktstellung der Windows-Plattform zurückgedrängt und die Abhängigkeit von einem Anbieter verringert werden. Im Folgenden wird zunächst kurz untersucht, wie die Wettbewerbssituation auf dem Softwaremarkt zu beurteilen ist. Anschließend wird überprüft, ob die Open Source-Förderung als Maßnahme der Wettbewerbspolitik geeignet ist.

Bei der Beurteilung des Wettbewerbs auf dem Softwaremarkt ist zunächst eine Marktabgrenzung erforderlich. Die Open Source-Förderung zielt hierbei auf die Marktposition von Windows auf dem Markt für Desktop-Betriebssysteme und die Marktposition des Office-Paketes auf dem Markt für Anwendungssoftware. Die Wettbewerbssituation auf dem Desktop- und Applikationenmarkt kann dabei durchaus unterschiedlich eingestuft werden. Gleiches gilt in noch stärkerem Maße für den

[153] Vgl. SCHMIDT UND SCHNITZER (2002), S. 27-28.

[154] Vgl. FRITSCH, WEIN, EVERS (2003), S. 83ff.

Server-Markt, auf dem verschiedene Produkte konkurrieren. Die Förderung des Einsatzes von Open Source-Software würde sich also nicht zielgerichtet gegen eine bestimmte Marktstellung in einem Teilmarkt richten, sondern hätte weitreichende Folgen auf für andere sehr wettbewerblich geprägte Märkte.[155]

Wie die Darstellung der Eigenschaften von Software gezeigt hat, unterliegt der Wettbewerb auf dem Softwaremarkt zudem besonderen Merkmalen. Diese fördern einerseits die Herausbildung einer bestimmten Marktstellung, andererseits begrenzen sie aber auch die Möglichkeiten den Preissetzungsspielraum auszunutzen.[156] Wie bereits gezeigt, kommt es auf dem Softwaremarkt häufig zu einem Wettbewerb um den Markt, statt zu einem Wettbewerb im Markt.

Fraglich ist, wie hoch die Markteintrittsbarrieren auf dem Softwaremarkt sind. Da die Entwicklungskosten für einen etablierten Anbieter versunkene Kosten darstellen, könnte er seinen Marktpreis so setzen, dass Wettbewerber vom Markteintritt abgehalten werden. Zudem unterstützen die zuvor beschriebenen Netzwerkeffekte insbesondere dann die Marktposition eines etablierten Anbieters, wenn hohe Wechselkosten bestehen. Allerdings gibt es auch verschiedene Faktoren, die die Höhe der Wechselkosten begrenzen. So könnte auf einem Markt, der durch einen stetigen Zustrom an Neukunden gekennzeichnet ist, ein Markteintritt auch im Wettbewerb um diese Kunden, die noch keine Wechselkosten haben, erfolgen. Denkbar wäre auch, dass „Netzwerkbrücken" gebaut werden, die eine Kompatibilität zwischen verschiedenen Plattformen herstellen.[157] Zum anderen bestehen auch für einen etablierten Anbieter Anreize, seine Produkte stetig weiterzuentwickeln, da aufgrund der bei Software fehlenden Abnutzung nach Erreichen der Marktsättigung nur dann Erträge realisieren kann, wenn durch neue Produkte Wiederholungskäufe initiiert werden.[158]

Die vorangehenden Punkte können an dieser Stelle nur andeuten, dass die Beurteilung der Wettbewerbssituation sehr differenziert erfolgen muss und keineswegs auf Basis allein des Marktanteils in einem Teilmarkt erfolgen kann. Sofern man dann tatsächlich eine Situation feststellt, die ein Eingreifen erforderlich macht, stellt

[155] Vgl. zu den wettbewerblichen Aspekten auch EVANS (2002), S. 44ff.

[156] Vgl. zu den Wettbewerbsüberlegungen auch SCHMIDT UND SCHNITZER (2002), S. 7.

[157] So erleichtert beispielsweise das Angebot des Microsoft Office Paketes auch für die Apple Plattform den Wechsel von Nutzern zwischen dem Windows- und Apple-Betriebssystem, da Dateien auf beiden Systemen verwendet werden können.

[158] Vgl. GRÖHN (1999), S. 140.

sich die Frage, ob die Förderung von Open Source-Software hierfür ein geeignetes Instrument ist.

Sofern ein Anbieter über eine marktmächtige Position verfügt, kennen das deutsche und europäische Wettbewerbsrecht eine Vielzahl an Maßnahmen und Institutionen, die gemäß den gesetzlichen Grundlagen in einem solchen Fall einschreiten. Nach einer Untersuchung der Wettbewerbssituation und der Feststellung eines wettbewerbswidrigen Verhaltens sind verschiedene Maßnahmen unterschiedlicher Eingriffsintensität möglich – die gezielte staatliche Förderung eines Wettbewerbers oder einer alternativen Technologie gehört allerdings nicht zu diesen Maßnahmen. Das deutsche und europäische Wettbewerbsrecht sehen keine aktiven Eingriffe zugunsten einer Marktseite vor, sondern setzen mit ihren Maßnahmen darauf, das wettbewerbswidrige Verhalten zu unterbinden und so den Marktkräften wieder den notwendigen Spielraum zu verschaffen. Die letztendliche Entscheidung über die Durchsetzung eines Anbieters kann dann der Marktprozess treffen. Die Förderung eines bestimmten Anbieters oder einer speziellen Technologie als wettbewerbspolitische Maßnahme einzusetzen, übersieht, dass staatliche Instanzen über die hierzu notwendigen Informationen gar nicht verfügen können. Der Schutz der Wettbewerbsordnung kann nur gelingen, wenn sie nach klaren und transparenten Regeln abläuft, die an der Ordnung, nicht am Marktergebnis ansetzen. Interventionen zugunsten bestimmter Technologien laufen aber – insbesondere aufgrund des Informationsdefizits staatlicher Stellen – geradezu zwangsläufig darauf hinaus, dass Private ermuntert werden, den Staat für ihre Partikularinteressen unter dem Deckmantel vermeintlicher Wettbewerbspolitik zu vereinnahmen.

Auch die aktuell und zuletzt diskutierten wettbewerbspolitischen Fragestellungen im Hinblick auf die Windows-Plattform beziehen sich nicht primär auf die Möglichkeiten der Etablierung einer zweiten Plattform. Aufgrund der beschriebenen Netzwerkeffekte kann es volkswirtschaftlich effizient sein, dass sich nur eine Plattform im Markt durchsetzt. Die wettbewerbspolitischen Ansatzpunkte sind vielmehr auf den Zugang anderer Anbieter zur Plattform bzw. auf die Ausdehnung ihrer Funktionalität und die Integration und enge Koppelung einzelner Anwendungen an diese Plattform gerichtet. Sofern an diesen Punkten ein wettbewerbsschädliches Verhalten vorliegt, sind keine Maßnahmen notwendig und angemessen, die eine komplette Ersetzung einer Plattform durch eine andere zum Ziel haben.

Häufig wird auch argumentiert, die Förderung einer alternativen Plattform verringere die Abhängigkeit von einem Anbieter und wirke „Monokulturen" entgegen.[159] Grundsätzlich ist es aus Nachfragersicht positiv zu beurteilen, wenn keine Abhängigkeit von speziellen Lieferanten besteht. Allerdings ist auch hier zu berücksichtigen, dass gerade auf dem Softwaremarkt die Entscheidung für verschiedene Plattformlieferanten auch eine Abschwächung der zuvor beschriebenen positiven Netzwerkeffekte bedeutet. Daher ist eine Abwägung der Vorteile durch eine geringere Abhängigkeit von einem Lieferanten mit den Nachteilen durch die Nutzung von verschiedenen Plattformen notwendig.

Fraglich ist auch, ob sich die Abhängigkeiten durch die Entscheidung für Open Source-Software tatsächlich verringern oder lediglich verschieben. Sofern der Nachfrager die Entscheidung für eine bestimmte Software getroffen hat, ist er wieder vom Anbieter dieser Software abhängig. Da für Open Source-Software die gleichen Netzwerkeffekte gelten wie für kommerzielle Software, können sich auch hierbei im Falle von Open Source-Software wieder Monopolsituationen aufgrund von Lock-in-Effekten herausbilden.

Im Falle von Open Source-Software entsteht aufgrund der besonderen Produktionssituation zwar keine Abhängigkeit von einem Produzenten, es ergibt sich jedoch eine Abhängigkeit entsprechend der gewählten Softwaredistribution. Je spezifischer das Wissen des Lieferanten um die speziellen Installationsbedingungen ist, desto größer werden auch in diesem Fall die Wechselkosten für den Nutzer sein. Alle Netzwerkeffekte und Wechselkosten, die im kommerziellen Bereich auftreten können, würden dann in gleicher Weise im Open Source-Bereich vorliegen. Die grundsätzlichen Abhängigkeiten lassen sich somit nicht verringern. Die Untersuchung der kommerziellen Open Source-Geschäftsmodelle hat zudem gezeigt, dass kommerzielle Anbieter Open Source nutzen, um die in Open Source getätigten Investitionen mit Erlösen aus komplementären Produkten und Diensten zu finanzieren. Schafft es ein kommerzieller Anbieter, sein Open Source-Angebot eng an dieses komplementäre Produkt zu koppeln – und dies ist notwendig, wenn er mit Open Source kommerziellen Erfolg haben will –, dann verschieben sich die Abhängigkeiten weg vom Open Source-Produkt hin zum komplementären Produkt.

[159] Vgl. Zypries (2001).

2.4.3.3 Keine standortpolitische Aufgabe - Open Source und der mittelständische Softwaremarkt

Die Förderung von Open Source-Software wird gelegentlich auch als standortpolitische Maßnahme angesehen. Durch die Nachfrageentscheidung des Staates für eine bestimmte Technologie soll dieser Technologie eine kritische Nachfragemenge verschafft werden, um so die heimischen Anbieter dieser Technologie zu unterstützen. Hiermit ist die Vorstellung verbunden, dass sich ein Sektor ohne diese Nachfrageentscheidung gar nicht oder nur schlechter entwickeln kann und der Staat die Marktentwicklung als Initialnachfrager fördern kann. Von der Förderung einer Technologie, die von lokalen Firmen entwickelt, installiert und gewartet wird, erwartet man zudem günstigere Effekte für die Wirtschaftsentwicklung im eigenen Land als von einer vollständig in einem anderen Land entwickelten Technologie. Im Folgenden wird daher untersucht, inwieweit der Staat geeignet ist, durch die Förderung von Open Source-Software einen positiven Effekt für die Wirtschaftsentwicklung im eigenen Land zu erreichen.

Die Umlenkung von Produktionsströmen in das eigene Land ist ein altes handelsstrategisches Argument. Es widerspricht grundsätzlich dem Prinzip der internationalen Arbeitsteilung und übersieht die tatsächlich auf dem Softwaremarkt schon heute gegebene Produktionsstruktur. Genauso wie eine Arbeitsteilung und Spezialisierung innerhalb einer Volkswirtschaft den gesamtwirtschaftlichen Wohlstand erhöht, so gilt dies auch für die internationale Arbeitsteilung. Gerade Deutschland als eine der wichtigsten Exportnationen weltweit profitiert besonders von dieser Arbeitsteilung. Aufgrund der horizontalen Struktur des Softwaremarktes sind zudem auch bei Software die aus dem Ausland importiert wird, große Teile der Wertschöpfungskette von inländischen Unternehmen abgedeckt. So werden beispielsweise allein bei den Certified Partnern von Microsoft in Deutschland 76.000 Microsoft-bezogene Arbeitsplätze geschaffen.[160]

Wie die Analyse in Abschnitt 2.3 gezeigt hat, basiert das Open Source-Modell im Kern auf einem bewusst marktfreien Koordinationsmechanismus. Dieser ist nicht nur hinsichtlich der Erfüllung der Marktfunktionen unterlegen. Erfolgt die Softwareerstellung marktfern, fallen in diesem Teil der Wertschöpfungskette auch keine volkswirtschaftlich relevanten Transaktionen an. Wird (bzw. muss) die Software

[160] Vgl. KOOTHS, LANGENFURTH UND KALWEY (2003). Siehe zur Bedeutung des kommerziellen IT-Sektors in verschiedenen Ländern Smith (2002), S. 79.

unentgeltlich abgegeben (werden), entstehen mit ihrer Entwicklung anders als auf dem kommerziellen Softwaremarkt auch keine Erlöse, Einkommen, Arbeitsplätze und Steuern.

Verzichtet die Open Source-Softwareentwicklung auf eine Bepreisung der erstellten Software, so können Entwickler auch nicht aus den Erträgen der erstellten Produkte bezahlt werden. Komplementärstrategien (Erzielung von Erlösen nicht mit der Software, sondern mit zusätzlichen Diensten oder Produkten) sind gesamtwirtschaftlich nicht geeignet, die Wertschöpfungsausfälle durch den Wegfall der Erlöse aus der Softwareentwicklung auszugleichen (siehe hierzu ebenfalls Abschnitt 2.3, insbesondere Gliederungspunkt 2.3.5.3). Im Open Source-Modell bringen die Entwickler zwar eine Leistung ein, dieser steht aber keine volkswirtschaftlich relevante Transaktion gegenüber. Wird kommerzielle Software durch Open Source-Produkte substituiert, so entfallen damit auch die Arbeitsplätze, die mit dieser kommerziellen Software verbunden sind. Software-Entwickler, die als Angestellte bei heimischen Softwareunternehmen Teil der volkswirtschaftlichen Wertschöpfungskette sind, werden so aus dem Markt gedrängt. Die Entwicklung von Open Source-Software vollzieht sich damit im Kern nicht nur außerhalb des kommerziellen Gütermarktes, sie vollzieht sich auch außerhalb des Arbeitsmarktes und hat so keinerlei positive, sondern insgesamt negative Effekte.

Die Softwareentwicklung ist nur dann auch im Open Source-Modell mit volkswirtschaftlich relevanten Transaktionen verbunden, wenn die Entwickler als Dienstleistung direkt für die Programmierung bezahlt werden (Erstellung von Individualsoftware). Diese unterscheidet sich aber nicht vom herkömmlichen kommerziellen Softwaremarkt und kann dann auch nicht die vom Open Source-Modell erhofften Vorteile (Open Source-Spirit) für sich in Anspruch nehmen.

In Deutschland steht die Softwareentwicklung gemäß Dienstleistungsstatistik für einen Umsatz von 19,4 Mrd. Euro, 8.806 Unternehmen und 131.356 Beschäftigte. 99,6% dieser Unternehmen gehören dabei dem IT-Mittelstand an. Der Wegfall eines Teils der Wertschöpfungskette und der damit verbundenen Erlösmöglichkeiten und Arbeitsplätze trifft somit insbesondere den IT-Mittelstand, denn die Open Source-Software bietet dem Mittelstand keine neuen, sondern nur einen Teil der bereits auf dem kommerziellen Softwaremarkt gegebenen Geschäftsmöglichkeiten. Diese wirken gesamtwirtschaftlich substitutiv und nicht additiv.

Wird ein mittelständischer Softwareanbieter bei der Programmierung von Standardsoftware vor die Frage gestellt, ob er für das Open Source-Modell oder für den

kommerziellen Markt entwickeln soll, so bietet ihm das kommerzielle Modell die Möglichkeit, seine Entwicklerinvestitionen auch über einen Verkauf der erstellten Programme wieder zu erlösen. Im Open Source-Modell kann er nur mit den an die Programmierung anschließenden Leistungen Erlöse erzielen. Je weiter sich das Open Source-Modell verbreitet, desto kleiner wird so der Bereich, auf dem Software kommerziell verwertet werden kann. Damit ist aus standortpolitischer Sicht eine Open Source-Unterstützung als Maßnahme zur Förderung des IT-Mittelstandes nicht nur ungeeignet. Sie ist schädlich, denn dem Softwaremarkt wird durch Open Source-Software die ökonomische Basis zur Realisierung von Erlösen und damit auch zur Schaffung von Arbeitsplätzen entzogen. Der deutsche IT-Markt ist funktionsfähig und hat insgesamt schon heute mit zwei Prozent der gesamten Wertschöpfung eine nicht zu unterschätzende volkswirtschaftliche Bedeutung. Open Source-Software führt zu einer allmählichen Aushöhlung dieses Marktes.

Literaturverzeichnis

BACKHAUS, K. (1999): Happy Engineering, in: managermagazin, Heft 8, 1999, S. 130-133.

BALZERT, H. (1996): Lehrbuch der Software-Technik: Software-Entwicklung, Heidelberg.

BERLECON RESEARCH (2002a): Free/Libre Open Source Software: Survey and Study, Use of Open Source-Software in Firms and Public Institutions Evidence from Germany, Sweden and UK, FLOSS Final Report - Part 1, Berlin, Juli 2002.

BERLECON RESEARCH (2002b): Free/Libre Open Source Software: Survey and Study, Firms' Open Source Activities: Motivations and Policy Implications, FLOSS Final Report - Part 2, Berlin, Juli 2002.

BERLECON RESEARCH (2002C): Free/Libre Open Source Software: Survey and Study, Basics of Open Source Software Markets and Business Models, FLOSS Final Report – Part 3, Berlin, Juli 2002.

BERLIOS (2003): http://openfacts.berlios.de/index.phtml?title=Open-Source-Lizenzen (27. Oktober 2003)

BITKOM (2003A): SoftwareTag 2003 Berlin-Brandenburg, Rahmenbedingungen für einen erfolgreichen IT-Mittelstand, Thomas Mosch.

BITKOM (2003B): ITK-Branche geht offensiv in die nächsten Jahre, Pressemitteilung BITKOM vom 23. September 2003.

BLANKART, C. B. UND KNIEPS, G. (1992): Netzökonomik, in: Jahrbuch für Neue Politische Ökonomie, Band 11, S. 72-87, Tübingen.

BORCHERT, M. UND GROSSEKETTLER, H. (1985): Preis- und Wettbewerbstheorie – Marktprozesse als analytisches Problem und ordnungspolitische Gestaltungsaufgabe, Stuttgart u.a.O.

BOSTON CONSULTING GROUP (2002): Hacker Survey, Release 0.3, Lakhani, K. R., Wolf, B. und Bates, J., LinuxWorld Präsentation, 31. Januar 2002.

BURR, W. (1995): Netzwettbewerb in der Telekommunikation: Chancen und Risiken aus Sicht der ökonomischen Theorie, Wiesbaden.

COLECCHIA, A. AND P. SCHREYER (2001): ICT Investment and Economic Growth in the 1990s: Is the United States a Unique Case? A Comparative Study of Nine OECD Countries, STI Working Paper 2001/7, OECD, Paris.

EITO (2003): European Information Technology Observatory 2003, 11. Ausgabe.

EVANS, D. UND REDDY, B. (2002): Government Preferences for Promoting Open Source Software: A Solution in Search of a Problem, National Economic Research Associates, Cambridge, Mass., April 2002.

EVANS, R. (2002): Politics and Programming: Government Preferences for Promoting Opne Source Software, in: Government Policy towards Open Source Software, Editor: Hahn, R. W., S. 34-49.

FEHR, E. UND SCHMIDT, K. M. (2001): Theories of fairness and reciprocity : evidence and economic applications, Münchener wirtschaftswissenschaftliche Beiträge.

FRANCK, E., JUNGWIRTH, C. (2002): Das Open-Source-Phänomen jenseits des Gift-Society-Mythos, In: Wirtschaftswissenschaftliches Studium : Wist, Band 31, 2002, Heft 3, S. 124-129.

FRITSCH, M., WEIN, T., EWERS, H.-J. (2003): Marktversagen und Wirtschaftspolitik, 5., überarbeitete und ergänzte Auflage.

GRASSMUCK, V. (2002): Freie Software zwischen Privat- und Gemeineigentum; Bundeszentrale für Politische Bildung, Bonn.

GRÖHN, A. (1999): Netzwerkeffekte und Wettbewerbspolitik - eine ökonomische Analyse des Softwaremarktes, Tübingen.

GROSSEKETTLER, H. (1991): Die Versorgung mit Kollektivgütern als ordnungspolitisches Problem, in: Ordo, Jahrbuch für die Ordnung von Wirtschaft und Gesellschaft, Band 42, S. 69-89, Stuttgart.

GROSSEKETTLER, H. (1995): Öffentliche Finanzen, in: D. Bender u.a. (Hrsg.), Vahlens Kompendium der Wirtschaftstheorie und Wirtschaftspolitik, Bd. 1, 6. Auflage, S. 483-627, München.

HABER, G. UND GETZNER, M. (2003): Gesamtwirtschaftliche Effekte des Softwaresektors in Österreich 2003, Forschungsbericht, Version 2.31, 5. September 2003, Universität Klagenfurt.

HANG, J. UND HOHENSOHN, H. (2003): Eine Einführung zum Open Source Konzept aus Sicht der wirtschaftlichen und rechtlichen Aspekte, eine Studie im Rahmen des Projektes NOW: Nutzung des Open Source Konzeptes in Wirtschaft und Industrie, Siemens Business Services GmbH & Co, OHG, C-Lab, Paderborn.

HOCH, D. J. ET AL. (1999): Secrets of Software Success: Management Insights from 100 Software Firms around the World. Boston, Massachusetts, 1999.

HORST, E. (2003): Unser Bürgermeister soll schöner werden, Die Stadt München will zum Betriebssystem Linux wechseln, in: Frankfurter Allgemeine Zeitung, 2. August 2003, S. 36.

IANSITI, M. UND LERNER, J. (2002): Evidence Regarding Microsoft and Innovation, AEI-Brookings Joint Center for Regulatory Studies, Related Publication 02-4, Washington, D.C.

INSTITUT FÜR MITTELSTANDSFORSCHUNG (2002A): Mittelstand in der Gesamtwirtschaft – Anstelle einer Definition, Brigitte Günterberg und Hans-Jürgen Wolter, Institut für Mittelstandsforschung, Bonn.

INSTITUT FÜR MITTELSTANDSFORSCHUNG (2002B): Unternehmensgrößenstruktur in Deutschland nach Wirtschaftsbereichen und Rechtsformen, Brigitte Günterberg und Hans-Jürgen Wolter, Institut für Mittelstandsforschung, Bonn.

JANKO, W. H., BERNROIDER, E. W. N. UND EBNER, W. (2000): Softwarestudie 2000, Eine empirische Untersuchung der österreichischen Softwarebranche.

JOHNSON, J. P. (2001): Economics of Open Source Software.

KATZ, M. L. UND SHAPIRO, C. (1994): Systems competition and network effects, in: The journal of economic perspectives. 8. Jahrgang, Heft 2, S. 93-115.

KLODT, H., LAASER, C.-F., LORZ, J. O., MAURER, R. (1995): Wettbewerb und Regulierung in der Telekommunikation, Kieler Studien 272, Tübingen.

KOOTHS, S., LANGENFURTH, M., KALWEY, N. (2003): Die Bedeutung der Microsoft Deutschland GmbH für den deutschen IT-Sektor (Economic Impact Study), MICE Economic Research Studies, Vol. 3.

KUHN, A. (2003): Die methodische Behandlung von Software in der Außenhandelsstatistik, in: Wirtschaft und Statistik, Heft 2/2003, S. 121-125.

LEE, S., MOISA, N. UND WEISS, M. (2003): Open source as a signalling device : an economic analysis, Johann-Wolfgang-Goethe-Univ., Fachbereich Wirtschaftswiss., Working paper series.

LEHRER, M. (2000): From factor of production to autonomous industry: the transformation of Germany's software sector, in: Vierteljahrshefte zur Wirtschaftsforschung. Berlin : Duncker & Humboldt, Bd. 69 (2000), 4, S. 587-600.

LERNER, J. UND TIROLE, J. (2000): The simple economics of open source, NBER working paper series, Nr. 7600.

LERNER, J. und Tirole, J. (2002): The scope of open source licencing, NBER working paper series, Nr. 9362.

LESSIG, L. (2002): Open Source Baselines: Compared to What? In: Government Policy towards Open Source Software, Editor: Hahn, R. W., S. 50-68.

LÜNENDONK (2003): Lünendonk-Liste I 2003: Die Top 25 Standard-Software-Unternehmen in Deutschland, Lünendonk GmbH, Bad Wörishofen.

MENDYS-KAMPHORST, E. (2002): Open vs. closed : some consequences of the open source movement for software markets, CPB discussion paper.

MENDYS-KAMPHORST, E. (2002): Open vs. closed: some consequences of the open source movement for software markets, CPB discussion paper.

MICROSOFT (2003): Creating a Vibrant Information Technology Sector: Growth, Opportunity and Partnership, White Paper, updated May 2003.

MORNER, M. (2003): Open Source Software, in: Das Wirtschaftsstudium: wisu, Bd. 32 (2003), H. 3, S. 318-321.

MUSTONEN, M. (2002): Why do firms support the development of subsitute copyleft programs? FPPE, Universität Helsinki.

O. V. (2002): Siehe hierzu: http://www.heise.de/newsticker/data/odi-27.02.02-000/

O. V. (2003): Microsoft at the power point, in: The Economist, N. 8341 (Sep. 13th – 19th 2003), S. 61.

OECD (2002): Information Technology Outlook, ICTs and the Information Economy.

OSTERLOH, M. KUSTER, B. UND ROTA, S. (2002): Die kommerzielle Nutzung von Open-Source-Software: der Einfluss von sozialem Kapital, in: Zeitschrift Führung + Organisation, 2002, Heft 4, S. 211-217.

RAHMEN-ZUREK, K. (2002): Internet economics and the organization of open source software-development In: Competition, environment and trade in the globalized economy / Dieckheuer, Gustav (2002), S. 101-132

RAYMOND, E. S. (1998A): The Cathedral and the Bazaar. Elektronische Ausgabe unter: www.openresources.com/documents/cathedral-bazaar/.

RAYMOND, E. S. (1999): The Magic Cauldron http://www.tuxedo.org/~esr/writings/magic-cauldron/.

RAYMOND, E.S. (1998B): Homesteading the Noosphere, Elektronische Ausgabe unter www.firstmonday.dk/issues/issue3_10/raymond/.

RÖLLER, L.-H. (2002): Der Charme der freien Software, in: Frankfurter Allgemeine Zeitung, 18.09.2002, Nr. 217, S. 15.

SCHIFF, A. (2002): The economics of open source software: A survey of the early literature, The Review of Network Economics Volume 1, Issue 1, S. 66-74.

SCHMIDT, K. UND SCHNITZER, M. (2003): Public subsidies for open source? : Some economic policy issues of the software market, Centre for Economic Policy Research, Discussion paper series.

SHAPIRO UND VARIAN (1999): Information rules: a strategic guide to the network economy.

SMITH, A. (1776): Veröffentlichung 2000: An inquiry into the nature and causes of the wealth of nations, Einführung von R. Reich Anmerkungen, Zusammenfassung und Index von E. Cannan, New York, Modern Library, 2000.

SMITH, B. L. (2002): The Future of Software: Enabling the Marketplace to Decide, in: Government Policy towards Open Source Software, Editor: Hahn, R. W., S. 69-86.

SPINDLER, G. (2003): Rechtsfragen der Open Source Software, im Auftrag des Verbandes der Softwareindustrie Deutschlands e. V.

STALLMAN, R. (2001): Free Software: Freedom and Cooperation, Abschrift der Rede an der New York University am 29. Mai 2001 http://www.gnu.org/events/rms-nyu-2001-transcript.txt

STATISTISCHES BUNDESAMT (1999): Klassifikation der Wirtschaftszweige mit Erläuterungen, Ausgabe 1999 (WZ93).

STATISTISCHES BUNDESAMT (2001): Umsatzsteuerstatistik, Fachserie 14, Reihe 8.

STATISTISCHES BUNDESAMT (2002A): Dienstleistungen in Deutschland, Ergebnisse der neuen Statistik - Jahr 2000, Wiesbaden.

STATISTISCHES BUNDESAMT (2002B): Volkswirtschaftliche Gesamtrechnung, Input-Output-Tabellen, 1991 bis 2000, erschienen im Juli 2002, Wiesbaden.

UNILOG INTEGRATA (2003): Client Studie der Landeshauptstadt München, Kurzfassung des Abschlussberichts, abgestimmte Fassung, 02.07.2003.

VANBERG (1997): Die normativen Grundlagen von Ordnungspolitik. In: ORDO, Bd. 48. S. 707-726.

WEBER, S. (2000): The Political Economy of Open Source, BRIE Working Paper 140, Economy Project Working Paper 15, Juni 2000.

WEIZSÄCKER, C. C. VON UND KNIEPS, G. (1989): Telekommunikation, in: OBERENDER, P. (Hrsg.) Marktökonomie: Marktstruktur und Wettbewerb in ausgewählten Branchen der Bundesrepublik Deutschland, München.

ZYPRIES, B. (2001): „Linux-Tag: Abhängigkeit von Softwareherstellern verringern" Artikel in der Zeitung „Die Welt" vom 06. Juli 2001, http://www.bmi.bund.de/top/dokumente/Rede/ix_47733.htm (27. Oktober 2003).