
# Jenaer Schriften zur Wirtschaftswissenschaft

## Volkswirtschaftliche Aspekte der Open-Source-Softwareentwicklung

*Markus Pasche, Sebastian von Engelhardt*

18/2004

**Arbeits- und Diskussionspapiere**

**der Wirtschaftswissenschaftlichen Fakultät**

**der Friedrich-Schiller-Universität Jena**

ISSN 1611-1311

**Herausgeber:**

Wirtschaftswissenschaftliche Fakultät
Friedrich-Schiller-Universität Jena
Carl-Zeiß-Str. 3, 07743 Jena

www.wiwi.uni-jena.de

**Schriftleitung:**

*Prof. Dr. Hans-Walter Lorenz*
h.w.lorenz@wiwi.uni-jena.de

*Prof. Dr. Armin Scholl*
a.scholl@wiwi.uni-jena.de

# Volkswirtschaftliche Aspekte der Open-Source-Softwareentwicklung

Markus Pasche, Sebastian v. Engelhardt

**Zusammenfassung:** Das Papier charakterisiert die spezifischen Eigenschaften des Gutes Software. Als Alternative zur Bereitstellung von Software über Märkte werden die Charakteristika der „freien“ Open-Source-Softwareentwicklung dargestellt, bei der die Entwickler auf eine unmittelbare kommerzielle Verwertung ihrer erstellten Leistung verzichten und den Quellcode öffentlich zugänglich machen. Der Schwerpunkt liegt dabei weniger auf einer ökonomischen Begründung solcher freiwilligen Beiträge zu einem Softwareprojekt, sondern auf den volkswirtschaftlichen Effizienzwirkungen dieser Produktionsweise. Zentrale Bedeutung wird den Aspekten der Humankapitalbildung und -allokation, der Rolle der Wissensakkumulation und der positiven Spillovereffekte von Wissen, sowie der Wirkung auf die Wettbewerbsintensität im kommerziellen Softwaresektor beigemessen. Ferner wird diskutiert, inwiefern das Open-Source-Modell gegenüber der Bereitstellung über den Markt Defizite bei der Durchsetzung von Konsumentenpräferenzen aufweist. Das Papier ist gleichzeitig ein kritischer Kommentar zur Open-Source-Software-Studie von Kalwey u. a. (2003b).

**Schlagworte:** Software, Open Source, Closed Source, Humankapital, Wissen, Netzwerkeffekte, Spillover, Effizienz, Verwertungsrechte, Marktunvollkommenheit.

**JEL-Klassifikation:** D62, D85, K11, L12, L13

Inhaltsverzeichnis

## Zusammenfassung zentraler Aussagen

1. Software ist ein Gut mit besonderen Eigenschaften. Sie ist ein *Erfahrungsgut*, dessen Eigenschaften zum Teil erst durch den Gebrauch vom Kunden erkannt werden. Der Anwender baut durch den Umgang mit dem Gut Humankapital auf. Erst die Verbindung von Software und Humankapital führt zu dem Kundennutzen. Weiterhin weist Software starke *Netzwerk-Externalitäten* auf, die zu der bekannten *Lock-In*-Problematik führen. Angebotsseitig ist die Softwareproduktion durch starke Skaleneffekte in Bezug auf die abgesetzten Lizenzen charakterisiert. Dies führt zu sehr engen oligopolistischen oder monopolistischen Marktstrukturen. Schließlich kann Software bzw. das im Quellcode enthaltene Wissen ein *öffentliches Gut* darstellen, wobei durch Kompilierung und Lizenzierung das Ausschlussprinzip wieder hergestellt werden kann, so dass Software zum *Club-Gut* wird. Dies ist die Voraussetzung für die Bereitstellung über Märkte.

2. Entwickler von Open-Source-Software (OSS) tragen in erheblichem Maß zu einem *Wissenskapitalstock* bei, der den Charakter eines *öffentlichen Gutes* hat. Der Schutz geistigen Eigentums und die Ausschließbarkeit durch kommerzielle Lizenzierungspolitik gelten nach gängiger Auffassung als notwendiger Anreizmechanismus zur privaten Produktion und Vermarktung von Software und somit von Wissen. Auf der anderen Seite werden gerade dadurch in erheblichem Umfang massive positive *Spillovereffekte* unterdrückt. Offensichtlich existieren alternative Mechanismen, bei denen gerade die Nutzung dieser Spillover-Effekte eine zentrale Rolle spielt.

3. Die Software- und somit Wissensproduktion im OSS-Sektor hängt stark von den Spillover- bzw. Netzwerkeffekten ab, die durch die Quelloffenheit und restriktive Lizenzen bedingt sind. Der Wissenskapitalstock trägt aber nicht nur zu Spillovereffekten *innerhalb* des OSS-Bereichs bei, sondern auch *zwischen* OSS- und dem kommerziellen Sektor.

4. Auf kommerziellen Märkten besteht der Preis einer Softwarelizenz aus einem Deckungsbeitrag für fixe Entwicklungskosten sowie aus einer Rente aufgrund monopolistischer Marktstrukturen. Bei OSS-Produkten entspricht der Marktpreis den Grenzkosten von Null. Die Entwicklerleistung wird gegebenenfalls durch die Vermarktung komplementärer Leistungen (teilweise) vergütet.

5. Die OSS-Produktion zieht zwar in geringem Maß Humankapital aus dem kommerziellen Sektor ab, so dass sich die Frage nach der *effizienten Allokation von Humankapital* stellt. In sehr viel höherem Maß schafft der OSS-Sektor aber *zusätzliches* Humankapital bzw. sorgt für dessen *bessere Auslastung* durch seine Fähigkeit, Programmierer (neben ihrer Berufstätigkeit) zu freiwilligen Kooperationsleistungen zu motivieren.

6. Komplexe Softwareprojekte erfordern Arbeitsteilung und Koordination vieler Entwickler. Der individuelle Entwicklungsbeitrag ist dabei zu gering bzw. zu schwer einzuschätzen, als dass es möglich wäre, dass *einzelne* Entwickler geistige Eigentumsrechte durchsetzen können. Im kommerziellen Sektor wird daher Software im Rahmen eines *Prinzipal-Agenten-Systems* entwickelt, wobei der Prinzipal (das Unternehmen bzw. das Management) die Verwertungsrechte hält. Im OSS-Bereich existiert kein Prinzipal. Die Motivation der Entwickler, freiwillige Entwicklungsbeiträge zu leisten, sind vielschichtig. Zum einen kann das Engagement als Investition in ein Softwareumfeld aufgefasst werden, welches die Basis für kommerzielle Komplementärprodukte darstellt. Intrinsische Motivation sowie Reputationseffekte (Erwerb von Kompetenzsignalen für den Arbeitsmarkt) können ebenfalls eine Rolle spielen.

7. Die OSS-Entwicklung trägt direkt und indirekt zu einer Erhöhung der Wettbewerbsintensität im kommerziellen Sektor bei und erhöht so dessen Effizienz, die aufgrund (produktionsbedingter) oligopolistischer und monopolistischer Strukturen Defizite aufweist.

8. In der Studie von Kalwey u. a. (2003b) wird die Kunden- bzw. Präferenzorientierung im kommerziellen Sektor überschätzt, da auf die Auswirkungen von Lock-In-Effekten (switching costs) sowie von monopolistischen Anbieterstrukturen nicht genügend eingegangen wird. Die Tendenz einer Entwickler- statt Kundenorientierung bei OSS-Projekten wird ebenfalls überschätzt, da Selektions- und Selbstbindungsmechanismen des OSS-Sektors nicht ausreichend gewürdigt werden. Zudem besteht aufgrund des Interesses an der Vermarktung von Komplementärprodukten ebenfalls ein Anreiz, eine breite Nutzerakzeptanz zu finden.

# 1 Einleitung

Der Begriff Open-Source-Software (OSS) und das Konzept „freier“ Softwareentwicklung, d.h. der weitgehende Verzicht auf unmittelbare kommerzielle Verwertung und stattdessen die Veröffentlichung des Programmcodes, existieren schon seit längerem. In den letzten Jahren tauchen solche Begriffe in der öffentlichen Debatte vor allem aus zwei Gründen zunehmend auf: Zum einen trifft OSS auch bei größeren Unternehmen sowie bei öffentlichen IT-Entscheidungsträgern verstärkt auf Akzeptanz[1]: Während sich OSS im Serverbereich schon seit längerem etabliert und dort auch maßgeblich die technische Entwicklung geprägt hat, bezieht sich diese Akzeptanz nun auch auf den Desktop-Bereich. Zum anderen wird kommerzielle Standardsoftware wegen immer neuer entdeckter Sicherheitslöcher sowie der Anfälligkeit gegen Viren, Würmer und Trojaner zunehmend kritisch betrachtet und hier ein komparativer Vorteil von OSS gesehen. Darüber hinaus findet das Konzept der „freien“ nicht-kommerziellen Softwareproduktion auch vor dem Hintergrund der aktuellen Debatte um den Schutz geistiger Eigentumsrechte und um Softwarepatente öffentliche Resonanz.

Im Folgenden werden OSS und „freie“ Software der Einfachheit halber synonym verwendet, auf die hinter dem Begriff „frei“ stehenden ethischen und gesellschaftstheoretischen Begründungen wird hier nicht weiter eingegangen (vgl. Stallman (2002)). Der Begriff OSS beschreibt Software, deren Quell- bzw. Programmcodes offen gelegt wird, d.h. prinzipiell jedem unentgeltlich zugänglich ist. Analog werden im Folgenden die Begriffe kommerzielle Software, proprietäre Software und Closed-Source-Software (CSS) als Synonyme behandelt. CSS meint in diesem Text vereinfacht alle Arten von Software, an der Eigentumsrechte definiert sind und somit eine kommerzielle Verwertung von Softwarelizenzen ermöglicht. Zum Schutz dieser Eigentums- und Verwertungsrechte wird der Programmcode grundsätzlich geheim gehalten (vgl. Kalwey u. a. (2003b), S.34, Lessig (2002), S.52 f., vgl. auch Grassmuck (2002)).

Die zunehmende Attraktivität von OSS hat dazu geführt, dass immer mehr kommerzielle Firmen explizite OSS-Strategien verfolgen, dieses Konzept zumindest aus Marketinggründen unterstützen, und teilweise auch selbst Entwicklungsbeiträge zu Open-Source-Projekten leisten[2]. Eine Reihe von Unternehmen bieten ausschließlich Komplementärprodukte zu OSS an, so dass deren Geschäftsmodell auf der Existenz freier Software beruht[3]. Kommerzielle Produkte und Dienstleistungen rund um OSS stellen bereits eine Wertschöpfung von mehreren Milliarden Dollar pro Jahr dar. So betrug in 2003 der Umsatz von Linux-basierten Servern, Software und Dienstleistungen bei IBM und HP rund 3,5 Mrd. US-Dollar (Heise News (2003), Heise News (2002b)). Dies sind gute Gründe, sich aus volkswirtschaftlicher Sicht mit dem Thema zu befassen, wobei mehrere Fragestellungen eine Rolle spielen:

[1] vgl. sueddeutsche.de (2003), Heise News (2002a), Heise News (2004d)

[2] Beispiele: IBM, HP, Sun, Intel als Hardware- bzw. Systemanbieter, Novell als Softwareanbieter.

[3] Beispiele: RedHat und SuSE als Linux-Distributoren, Ximian/Novell.

- Welche spezifischen Eigenschaften weist das Gut Software auf und welche Bedeutung hat dies für die Bewertung von OSS und CSS?
- Weshalb existieren überhaupt freiwillige Kooperationsleistungen, also Beiträge zur OSS-Entwicklung ohne kommerzielle Verwertungsabsicht?
- Welche Wechselbeziehungen bestehen zwischen kommerzieller und freier Softwareproduktion unter den Gesichtspunkten der Humankapital-Allokation, den positiven Wissens-Spillovern, den Netzwerk-Effekten sowie möglichen Wirkungen auf den Wettbewerb auf Softwaremärkten?
- Wie ist OSS unter Effizienz- bzw. Wohlfahrtswirkungen zu beurteilen?

Bezugspunkt dieses Papiers ist eine Studie des Münster Institute for Computational Economics (MICE). In dieser, von Microsoft Deutschland in Auftrag gegebenen Studie mit dem Titel „Open Source-Software: Eine volkswirtschaftliche Bewertung“, kommen die Autoren insgesamt zu einer negativen Bewertung von OSS (Kalwey u. a. (2003b)). Die wichtigsten methodischen Mängel dieser Studie liegen zum einen darin, dass die OSS-Entwicklung einem kommerziellen Softwaremarkt gegenüber gestellt wird, welcher als ein allein preisgesteuerter, kompetitiver und hocheffizienter Markt dargestellt wird, wie er zwar in Lehrbüchern zu finden ist, jedoch mit den angebots- und nachfrageseitigen Charakteristika von Software wenig zu tun hat. Durch den Vergleich von OSS mit den Eigenschaften eines perfekten, jedoch fiktiven Marktmodells ist die „Diagnose“ der MICE-Studie wenig überraschend, aber auch wenig relevant. Zum anderen wird kaum auf die Spillover-Effekte und Öffentliche-Gut-Eigenschaften des in Software codierten Wissens eingegangen, und auch die Wechselwirkungen zwischen kommerzieller und freier Softwareentwicklung wird nur unzureichend analysiert. Schließlich wird OSS in der Studie überwiegend als Resultat hobbyistischer Tätigkeit dargestellt und so die Anreiz-, Koordinations- und Kontrollmechanismen im OSS-Sektor nicht ausreichend gewürdigt.

Das Papier ist wie folgt aufgebaut: Im zweiten Kapitel werden wichtige Eigenschaften des Gutes Software charakterisiert, welche Implikationen für die Struktur eines kommerziellen Softwaremarktes, aber auch für die Eigenschaften „freier“ Softwareproduktion in vernetzten, selbstorganisierten Gruppen – im Folgenden „Community“ genannt – aufweisen. Im dritten Kapitel werden beide Bereitstellungsformen – kommerzieller Markt und Community – geschildert und miteinander verglichen. Das vierte Kapitel diskutiert volkswirtschaftliche Wirkungen der OSS-Entwicklung im Vergleich zur Bereitstellung über den Markt unter dem Gesichtspunkt der Effizienz. Dabei spielen Fragen der Humankapitalakkumulation und -allokation, der Wissens-Spillover, der Konsumentensouveränität und der Wettbewerbswirkungen eine zentrale Rolle. Das fünfte Kapitel greift die zusammenfassenden Thesen der MICE-Studie auf und kontrastiert diese mit den vorliegenden Ergebnissen.

## 2 Eigenschaften von Software

Das Gut Software weist einige spezifische Charakteristika auf, die es von klassischen privaten Gütern ebenso unterscheidet wie von klassischen öffentlichen Gütern. Die wichtigsten Eigenschaften dieses immateriellen, alle Lebensbereiche durchziehenden Gutes, sowie die sich daraus ergebenden Konsequenzen werden kurz dargestellt.

### *2.1 Software als Erfahrungsgut im doppelten Sinn*

Nachfrageseitig gesehen ist Software – wie alle Informationsgüter (vgl. Kotkamp (2001), S.58 ff.) – ein *Erfahrungsgut* (vgl. Nelson (1970)), und zwar im doppelten Sinne: Erstens können vom Konsumenten die Eigenschaften und Problemlösungsfähigkeiten des Produktes vor dem Kauf nur teilweise festgestellt werden. Das vollständige Wissen um den tatsächlichen Nutzen des Produktes erwirbt der Anwender also erst durch den Umgang mit der bereits erworbenen Software. Die Kaufentscheidung bei Software ist somit zum Teil stets eine *Entscheidung unter Unsicherheit*. Dies ist die Eigenschaft als Erfahrungsgut im herkömmlichen Sinne (vgl. Ewers u. a. (2003), S.286).

Zweitens wird durch den Umgang mit der Software *spezifisches Humankapital* aufgebaut, der Nutzer wird also zum *erfahrenen* Nutzer. Durch den täglichen Umgang mit der Software lernt der Nutzer diese immer besser kennen und erweitert seine Fähigkeiten. Die Produktivität im Umgang mit der Software steigt mit dem Humankapital. Völlig ohne Erfahrungen oder Wissen kann Software sogar nutzlos sein. Dies verdeutlicht, dass der Nutzen erst aus dem Zusammenspiel des Softwareproduktes mit Humankapital generiert wird[4]. Software unterscheidet sich insofern nicht von anderen Werkzeugen oder Maschinen, denn auch hier wird durch den täglichen Umgang – durch die Erfahrung („learning by doing“) – Humankapital aufgebaut (vgl. Arrow (1962)).

Humankapital kann in Form einer spezifischen oder eher allgemeinen Kompetenz aufgebaut werden, d.h. es werden spezielle Kenntnisse der Software XY, allgemeine Kenntnisse mit „Bürosoftware“ oder generelle Fähigkeiten im Umgang mit einem Betriebssystem bzw. mit Computern aufgebaut. Die Bedeutung dieses Software-spezifischen Humankapitals zeigt sich auch in entsprechenden Hinweisen in Stellenanzeigen. Es haben sich eine Reihe von Signalen entwickelt, mit denen solches Humankapital verbrieft sein soll. Man denke hier beispielsweise an den „Internetführerschein“ oder an diverse Qualifizierungskurse, an deren Ende die Teilnehmer entsprechende Zertifikate ausgehändigt bekommen.

[4] Dies ist ein illustratives Beispiel für Beckers Theorie der Haushaltsproduktion, vgl. Becker und Michael (1973).

### *2.2 Software als Netzwerkgut*

Neben dem direkten Zusammenspiel zwischen *einem* Nutzer und *seiner* Software existieren auf der Nachfrageseite auch Effekte zwischen *allen* Nutzern *einer* Software, also *Netzwerkeffekte* bzw. *Netzwerkexternalitäten* (vgl. Blankart und Knieps (1992), Katz und Shapiro (1994), Liebowitz und Margolis (1998)). Der Nutzen der Software wird größer, je mehr installierte Einheiten es gibt oder je größer der Marktanteil der Software ist. Dies erklärt sich aus dem leichteren Erfahrungsaustausch und den Vorteilen der Datenkompatibilität, sowie aus der erhöhten Chance, dass Komplementärprodukte entwickelt werden, was für erfolgreiche Software eher wahrscheinlich ist als für Nischenprodukte. Angebotsseitig existieren ebenfalls *Netzwerkexternalitäten* (vgl. Kalwey u. a. (2003b), S.21): Die Entwicklung von Software ist umso einfacher, je homogener das Umfeld dafür ist (z.B. einheitliches Design, gemeinsame Bibliotheken, Standard-Entwicklungstools, Standard-Schnittstellen, geringeres Marktrisiko, weil die Zahl der Update-Willigen groß und relativ stabil ist).

Liegen Netzwerkeffekte vor, so ist in der Regel eine *Standardisierung* wohlfahrtsmaximierend. Der Markt findet jedoch nicht in allen Fällen zur Standardisierung bzw. nur zu einer suboptimalen (d.h. zu kleinen) Netzwerkgröße. Zudem kann es zu einer inferioren Standardisierung, also zur Dominanz einer inferioren Technologie kommen (vgl. Gandal (1995), Gröhn (1999), S.48ff.). So können die Netzwerkeffekte vor allem auf der Nachfrageseite durch *Pfadabhängigkeiten* und *Lock-In-Effekten* zu stabilen, jedoch pareto-ineffizienten Gleichgewichten führen (Liebowitz und Margolis (1994), Gröhn (1999), S.39 ff.).

Es gibt vor allem drei Hemmnisse, die einen Wechsel zu einer anderen Software bzw. Technologie behindern und somit ein Verharren in einem ineffizienten Zustand begründen:

- Es gibt direkte Kosten des Wechsels (switching costs), die in dem Kauf neuer Lizenzen, Umrüstkosten (z.B. Konvertierung von Datenbeständen) und ggf. dem Kauf neuer Hardwarekomponenten im Falle fehlender Treiber bestehen.
- Bei einem unilateralem Wechsel zu einer konkurrierenden Software kommt es zu einem Verzicht auf die positiven Netzwerk-Externalitäten der bisherigen Softwarelösung. Diese stellen Opportunitätskosten dar.
- Wegen der Erfahrungsgut-Eigenschaft hat der Nutzer spezifisches Humankapital gebildet, welches bei einem Wechsel zu einem anderen Produkt zum Teil verloren geht und neue Humankapitalinvestitionen notwendig macht. Die Humankapitalabschreibungen stellen ebenfalls Opportunitätskosten dar, während das Sammeln neuer Erfahrungen ähnlich wie Umrüstkosten zu behandeln sind.

Klemperer (1995) zeigt, dass schon allein die Existenz von Wechselkosten zu Effizienzverlusten führt. Bei Software, wie bei allen Gütern mit Netzwerkeffekten, tritt also eine Tendenz zur Standardisierung auf. Da bei CSS die Dateiformate und Schnitttstellen sowie die interne Programmiertechnik in der Regel einen proprietären, d.h. nicht-offenen Standard darstellen, impliziert eine Standardisierung gleichzeitig auch die Monopolisierung des Marktes zugunsten des führenden Softwareanbieters. Während die Standardisierung einer Technologie an sich wohlfahrtsökonomisch positiv zu beurteilen ist, ist die damit verbundene Dominanz eines Anbieters aufgrund seiner Marktmacht problematisch. Neben dem statischen Wohlfahrtsverlust eines solchen Monopols kommt hier noch ein dynamischer Aspekt hinzu: Ein dominierender Anbieter kann auf verschiedene Weise versuchen, die switching costs in die Höhe zu treiben und damit den Lock-In-Effekt und somit seine Marktposition zu stärken. Dies kann z.B. durch die Etablierung proprietärer Schnittstellen und Dateiformate geschehen, die zu hohen Umrüstkosten (evtl. sogar zur Unbrauchbarkeit bisheriger Datenbestände) führen können. Ferner kann er bestimmten Nutzergruppen starke Vergünstigungen einräumen bzw. Lizenzen nahezu kostenlos überlassen, damit diese Nutzer spezifisches Humankapital aufbauen, welches ihre zukünftigen switching costs erhöht und sie zu künftigen zahlungsbereiten Kunden macht. Dafür lassen sich leicht Beispiele aus der Praxis finden. Ferner kann ein dominierender Anbieter dazu übergehen, statt einzelner Softwareprodukte ganze Bündel von Software zu vermarkten. Auf diese Weise kann er seine Marktmacht in einem Marktsegment dazu nutzen, auch in anderen Marktsegmenten Marktführer zu werden (sog. „leveraging“). Das Verhalten Microsofts, den Internet Explorer als integralen Bestandteil des Betriebssystems zu deklarieren oder die Multimediafähigkeit des Betriebssystems an die Benutzung des mitgelieferten Mediaplayers zu koppeln, sind solche Strategien, die u.a. vom Europäischen Gerichtshof als Missbrauch der marktbeherrschenden Stellung gewertet und mit Bußgeld belegt wurden. Zu der Frage der Marktmacht bzw. der monopolähnlichen Marktstellung von Microsoft hat sich inzwischen eine eigene Literatur entwickelt (vgl. unter anderem Fichert (1998), Gilbert und Katz (2001), Choi u. a. (1997)).

Hier könnte die Entwicklung offener Standards und Datenformate – auch wenn dies selbst ein solcher pfadabhängiger Prozess ist – helfen, Marktmacht einzelner Akteure zu verhindern, indem switching costs deutlich reduziert werden und somit eine gewisse Wahlfreiheit der Konsumenten hergestellt wird (vgl. Kuhlmann (2004)).

### *2.3 Skaleneffekte und Verbundvorteile*

Die Kostenfunktion bei der Produktion von Software ist gekennzeichnet von sehr hohen Entwicklungskosten, den sogenannten „first-copy-costs“, in Kombination mit sehr geringen Grenzkosten (vgl. Kalwey u. a. (2003b), S.20). Wird eine neue Software erstellt, dann fallen (versunkene) Kosten für die Entwicklung, Programmierung, Testdurchläufe, usw. an. Die fertige Software liegt dann in digitaler Form vor und kann beliebig

oft kopiert werden. Die nun auftretenden (Stück-)Kosten – für den Kopiervorgang, das Trägermaterial (z.B. DVD), die Verpackung und den Vertrieb – sind dann relativ gering, und gehen sogar gegen Null, wenn die Software auf Servern im Internet zum Download angeboten wird.

Die Softwareindustrie ist daher von degressiven Durchschnittskostenverläufen und damit tendenziell von Größenvorteilen bzw. Skaleneffekten geprägt. Zusätzlich treten noch Verbundvorteile in der Produktion auf, da bereits geschriebene Programmteile bei der Entwicklung anderer Software verwendet werden können (vgl. Gröhn (1999), S.5). Treten bei der Produktion eines Gutes Skaleneffekte und/oder Verbundvorteile auf, und sind diese hinreichend stark ausgeprägt, so handelt es sich hierbei um ein so genanntes natürliches Monopol: Diese Marktform liegt dann vor, wenn es am effizientesten ist, dass nur ein Unternemen ein Produkt bzw. eine Produktpalette anbietet, weil die Kostenfunktion subadditiv ist (vgl. Windisch (1987), S.41). Unterstellt man für alle Unternehmen die gleiche Kostenfunktion und identische Durchschnitts- und Grenzkostenverläufe, wird das Effizienzargument auf ein Kostenargument reduziert: natürliche Monopole sind dadurch gekennzeichnet, dass ein Unternehmen den Markt kostengünstiger bedienen kann als mehrere Anbieter (vgl. Knieps (2001), S.21).

Zwar wird die Softwareproduktion auch durch Skaleneffekte geprägt, jedoch sind diese nicht stark genug ausgeprägt, um ein natürliches Monopol anzunehmen. Hingegen werden die Monopolisierungstendenzen aufgrund der nachfrageseitigen Netzwerkeffekte als bedeutender angesehen (vgl. Fichert (2002), S.3). So lässt sich empirisch zeigen, dass die Konzentration auf Softwaremärkten vor allem auf Netzwerkeffekte zurückzuführen ist (vgl. Gröhn (1999), S. 112ff.). Dieses Ergebnis wird auch durch folgende Überlegung gestützt: Wenn die Marktmacht allein auf Skaleneffekte zurückzuführen wäre, kann erwartet werden, dass Konkurrenten über Produktdifferenzierung in den Markt eindringen. Eine solche Strategie ist jedoch bei starken Netzwerkeffekten nicht erfolgreich. Auch wenn Netzwerkeffekte als bedeutender angesehen werden, so gilt dennoch, dass die Eigenschaften der Produktionskostenfuktion von Software in der Tendenz eher die Herausbildung von Märkten mit wenigen großen, evtl. sogar mit einzelnen dominierenden Anbietern als eine Entwicklung atomistischer Märkte erwarten lassen.

### *2.4 Das Wissensgut Software – ein öffentliches Gut?*

Der entscheidende Produktionsfaktor von Software ist *Wissen*. Software selbst enthält wiederum ein hohes Maß an *codiertem Wissen* als *Output*. Die Verwendung einer Software durch einen Benutzer wird nicht durch die Verwendung derselben Software (in Form einer Kopie) durch einen anderen Nutzer eingeschränkt (vgl. Kalwey u. a. (2003b), S.22). Eine einmal erstellte Software stellt sozusagen eine *nicht* knappe Ressource dar, lediglich das Trägermaterial (z.B. CD-ROM) kann Knappheitsrestriktionen unterliegen. Wegen dieser Nichtrivalität beim Konsum und zunächst fehlender Ausschließbarkeit hat das in der Software steckende Wissen den Charakter eines *öffentli-*

*chen Gutes*. Die fehlende Ausschließbarkeit löst das bekannte *Trittbrettfahrerproblem* aus, was eine Bereitstellung solcher Güter durch den Markt äußerst schwierig macht. Programmierer haben demnach solange keinen Anreiz zur Produktion von Software, wie sie keine Verfügungsrechte an den von ihnen geschaffenen Werten durchsetzen, d.h. am Markt tauschen können. Entsprechend wird die Bereitstellung öffentlicher Güter auch als das klassische Feld öffentlicher (d.h. staatlicher) Aufgaben angesehen. Eine mögliche Lösung des Anreizproblems besteht darin, das Gut im staatlichen Auftrag zu produzieren, durch Steuern zu finanzieren und zu einem Preis von Null (entsprechend der Grenzkosten) abzugeben. Dies entspricht etwa der Lösung, wie sie für das Gut des universitären Wissens und der Bildung etabliert wurde. Eine andere Lösung des Problems besteht darin, eine Ausschließbarkeit von der Nutzung des codierten Wissens durch *property rights* sicherzustellen, also das Ausschlussprinzip wieder in Kraft zu setzen. So wird Software zu einem *Club-Gut* und der Anreiz zur Bereitstellung der Produkte über den Markt durch den Verkauf von Lizenzen ist wieder hergestellt (vgl. Kalwey u.a. (2003b), S.27).

Beide Lösungen weisen Nachteile auf (vgl. Quah (2003), S. 19ff, 26ff.): Im ersten Fall ist das wohlfahrtsoptimale Niveau der staatlichen Aktivität kaum zu ermitteln. Dies liegt zum einen an der schwierigen Ermittlung der Präferenzen und somit Zahlungsbereitschaften für das Gut, zum anderen daran, dass bei der Produktion neuen Wissens a priori nicht klar ist, ob der Produktionsprozess überhaupt zu einem Erfolg wird und wie das Ergebnis des Prozesses beurteilt werden kann. Auf der anderen Seite hat das Verfahren den Vorzug, dass produziertes Wissen öffentlich und kostenlos zugänglich ist und somit ein Maximum an positiven Externalitäten bzw. Spillover-Effekten erzeugen kann. Bei der zweiten Lösung verhält es sich genau umgekehrt: Das Niveau der Produktion wird über den Markt gesteuert und die Zahlungsbereitschaften werden durch die Bildung von Marktpreisen offenbart. Auf der anderen Seite wird das in Software codierte und per Lizenz genutzte Wissen geheim gehalten und positive Externalitäten somit unterdrückt. Dies stellt eine noch drastischere Betonung des Ausschlussprinzips dar, als beispielsweise die Vergabe von Patenten: Der *Patentschutz* ist die Lösung eines ähnlichen Anreizproblems, welches sich bei Forschung und Entwicklung ergibt. Um einen Anreiz für Forschung und Entwicklung zu geben, dessen Output ebenfalls Charakteristika eines öffentlichen Gutes aufweisen kann, werden Patente vergeben, d.h. es wird eine temporäre Monopolstellung mit entsprechenden Monopolrenten garantiert. Der temporäre (!) rechtliche Schutz der komerziellen Verwertung ist dabei bewusst verknüpft mit der Offenlegung des erzeugten Wissens durch die Beschreibung in der allgemein zugänglichen Patentschrift. Über Lizenzen vertriebene Closed-Source-Software hingegen verhindert nicht nur die Wissensverbreitung (keine Offenlegung), sondern stellt auch Monopolrenten für einen unbegrenzten Zeitraum sicher (keine zeitliche Beschränkung der Verfügungsrechte).

Das zugrunde liegende Problem (Dilemma) ist ein Konflikt zwischen der Notwendigkeit eines Anreizes zur Produktion des Wissens durch *Zugangsbeschränkung* und der

positiven Effizienzwirkung eines *freien* Zugangs zum Wissen – sowohl wegen der Nichtrivalität beim Konsum, als auch aufgrund der Spillovereffekte. Kommerzielle Software und OSS beschreiten in dieser Konfliktlage zwei völlig unterschiedliche Wege.

## 3 Formen der Bereitstellung

Die Bereitstellung von Software kann also entweder als *Club-Gut* über den Markt erfolgen, oder Software wird – in diesem Fall durch die OSS-Community – in Form eines *öffentlichen Gutes* bereitgestellt. Beide Möglichkeiten werden im Folgenden in ihrer Grundstruktur dargestellt. Eine abwägende Bewertung beider Bereitstellungsformen erfolgt dann anschließend durch den Vergleich von OSS und CSS unter Effizienzgesichtspunkten.

### *3.1 Bereitstellung über den Markt*

Um eine Bereitstellung über den Markt zu gewährleisten müssen durchsetzbare Verfügungsrechte definiert sein. Diese Ausschließbarkeit der Nutzung des codierten Wissens wird in zwei Schritten sichergestellt:

1. Anders als bei OSS wird bei CSS nicht der vom Menschen lesbare Quellcode vertrieben bzw. offengelegt, sondern lediglich das nur maschinenlesbare *binäre Kompilat*. Verkauft wird also lediglich das fertige Softwareprodukt, aber nicht das programmiertechnische Wissen, welches zu dieser Lösung führt. Das Verfügungsrecht über den Quellcode wird durch den technischen Vorgang des Kompilierens sehr effektiv geschützt. Die Empfänger des binären Codes sind von der Nutzung des Quellcodes ausgeschlossen, denn aus dem binären Code sind keine Rückschlüsse auf die Struktur und Programmierung der Software möglich (Kalwey u. a. (2003b), S.17 f.).

2. Auch das Kompilat, der binäre Code, ist mit Verfügungsrechten ausgestattet. Das Recht zur Nutzung einer Software erhält der Konsument meist durch den Erwerb einer entsprechenden *Lizenz*. Zur Durchsetzung des Ausschlussprinzips bedient man sich hier in aller Regel einer Kombination aus technischem und juristischem Kopierschutz (vgl. Kotkamp (2001), S.53).

Durch diese beiden Schritte wird das Ausschlussprinzip wieder in Kraft gesetzt und der Anreiz zur Bereitstellung der Produkte über den Markt ist wieder hergestellt. Allerdings ist zu bedenken, dass die diskutierten besonderen Eigenschaften des Club-Gutes Software Konsequenzen für die Marktstruktur und die Effizienz der Marktallokation haben.

Für den technischen Entwicklungsprozess gilt, dass komplexe Softwareprojekte nur in hochgradiger *Arbeitsteilung* entwickelt werden können, was wiederum koordinierende und steuernde Mechanismen erfordert. In einem kommerziellen Softwareunternehmen wird Software in einer *Prinzipal-Agenten-Struktur* entwickelt. Die Unternehmensleitung (Prinzipal) definiert gewisse Zielvorgaben und stellt Teams von Programmierern, Testern usw. (Agenten) zusammen, deren Tätigkeit vom Prinzipal nur sehr bedingt beobachtet und beurteilt werden kann. Es kann davon ausgegangen werden, dass die entsprechenden Arbeitsverträge anreizkompatibel ausgestaltet werden. Da in komplexen Projekten der Entwicklungsbeitrag des Einzelnen schwer definierbar ist, ist es kaum möglich, dass jeder Entwickler individuelle property rights an seinem Beitrag hält. Daher erhält der Prinzipal die Eigentumsrechte am ganzen Softwareprojekt und kann ggf. die Entwickler am Markterfolg beteiligen. Die Abtretung der Eigentumsansprüche am entwickelten Code ist in der Regel Teil des Arbeitsvertrages zwischen Prinzipal und Agent.

Das im Entwicklungsprozess eingesetzte Wissen entfaltet Spillover-Wirkungen lediglich innerhalb des Entwicklungsteams sowie bei ausgewählten Gruppen von Testern, denen der Sourcecode zugänglich gemacht wird. Da bei Open-Soure-Software weltweit jeder Zugang zum Code hat, ist die Wahrscheinlichkeit, dass Fehler entdeckt und gemeldet werden, hier deutlich höher. Aus diesem Grund versuchen kommerzielle Firmen zunehmend, diesen Vorzug der Community-Produktion durch Einbeziehung vertrauenswürdiger und zur Geheimhaltung verpflichteter Nutzergruppen in den Prozess der Tests und Pflege des Codes nachzuahmen (z.B. „shared source“ Programm bei Microsoft, vgl. Heise News (2004c)).

Der Vertrieb der Lizenzen erfolgt auf sehr unterschiedlichen Wegen. Im Fall privater Anwender können Lizenzen direkt im Einzelhandel erworben werden. Alternativ werden diese als OEM-Lizenz (Original Equipment Manufacturer) z.B. an Hardwareanbieter bzw. -einzelhändler verkauft, welche die Software auf den zu verkaufenden Computern vorinstallieren, oder sie werden als Campus-Lizenz an Schulen und Universitäten vertrieben. In diesen Fällen werden wenig oder keine Komplementärprodukte wie Support nachgefragt. Bei Firmenkunden erfolgt der Vertrieb oft über IT-Dienstleister. Solche Kunden fragen letztlich nicht „Software“ als Gut nach, sondern „Problemlösungen“, bestehend aus Softwarelizenzen, Beratung, individueller Konfiguration, Schulung, Pflege usw. Prinzipiell benötigt zwar jeder Nutzer eine „Problemlösung“, der Heimanwender baut sich jedoch das dafür notwendige Humankapital z.B. durch Produkterfahrung oder Literaturstudium selbst auf.

Insgesamt kann festgehalten werden, dass der arbeitsteilige Produktionsprozess sich in Gestalt einer Prinzipal-Agenten-Struktur vollzieht und die Ressourcen (insbesondere das Humankapital) weitgehend nach administrativen Vorgaben des markt- und gewinnorientierten Prinzipals gezielt und strategisch eingesetzt werden. Anreize zur Softwareproduktion und zur Innovation sind gegeben. Der Entwicklungsprozess ist, trotz kreativer Freiräume für einige Entwickler, zielgerichtet und eine Konsistenz der

Teilprojekte ist zumindest im Prinzip stets sichergestellt. Allerdings muss einschränkend gesagt werden, dass aufgrund der starken Informationsasymmetrien zwischen Prinzipal und Agenten sowie zwischen Agtentengruppen eine Beurteilung der Effizienz des Produktionsprozesses bzw. deren Sicherstellung nicht ganz unproblematisch ist. Nachteilig ist die geringe „Effizienz ex post“ aufgrund hoher Lizenzpreise und systematischer Marktversagensgründen wegen der Netzwerk-bedingten Lock-In-Effekten und dem strategischen Anreiz des Unternehmens, diese zu erhöhen oder zumindest zu stabilisieren. Außerdem besteht ein Problem darin, dass wegen der Geheimhaltung des Codes kaum Beiträge zu einem öffentlichen Wissenskapitalstock geleistet werden, der massive positive Externalitäten zur Folge hätte.

Für den Kunden besteht zum einen der Vorzug, dass der Entwicklungsprozess kontinuierlich und zielgerichtet verläuft und er somit ein hohes Maß an Sicherheit bezüglich seiner IT-Investitionen hat. Außerdem ist ein ständiger Support, also eine permanent möglich Inanspruchnahme von (humankapitalintensiven) Komplementärleistungen sichergestellt. Nachteilig sind für ihn neben dem hohen Lizenzpreis die einseitige Abhängigkeit von einem Unternehmen durch hohe (und strategisch weiter erhöhte) Wechselkosten sowie die geringere Transparenz und Flexibilität, da er selbst den Code nicht einsehen und ggf. modifizieren kann und sich in jeder Hinsicht auf die Angaben des Herstellers verlassen muss. Somit besteht eine Informationsasymmetrie, die sich negativ auf den Kunden auswirken kann (z.B. die Möglichkeit, dass in der Software sog. „backdoors“ enthalten sind, die einen unberechtigten Datenzugriff ermöglichen).

### *3.2 Bereitstellung durch die Open-Source Community*

Wird bei der Bereitstellung von Software der Charakter eines öffentlichen Gutes betont, so soll gerade *kein* Ausschlussprinzip durchgesetzt, sondern die allgemeine, unentgeltliche Verwendung und Verbreitung gewährleistet werden. So ist festzustellen, dass es für OSS unterschiedliche Lizenzmodelle gibt und kaum eine Software ohne jeglichen Schutz bzw. Nutzungsbestimmungen entwickelt und freigegeben wird. Am bekanntesten ist die GNU General Public License (GPL), bei der die Nutzung, Modifikation und Weitergabe des Gutes erlaubt, die kommerzielle Verwertung des Codes innerhalb proprietärer Software jedoch untersagt ist (vgl. www.gnu.org). Es existieren aber auch Lizenzmodelle, bei denen der Verkauf des Codes oder dessen Verarbeitung in proprietärer Software nicht gänzlich untersagt ist (vgl. Koglin und Metzger (2004), Mustonen (2003)). Ferner gibt es auch Beispiele dafür, dass der Code parallel unter mehreren Lizenvereinbarungen verbreitet wird. Im Folgenden wird jedoch von restriktiven Lizenzmodellen ähnlich der GPL ausgegangen. Bei solchen Lizenzen kann es allenfalls zu freiwilligen Zahlungen (Schenkungen, Sponsoring) kommen, oder es werden zusätzliche Dienstleistungen (Konfigurieren, Vorkompilieren etc.) vergütet.

Es stellt sich die Frage nach dem Anreizsystem bzw. der Motivation der Entwickler, die erklärt, weshalb sie freiwillige Beiträge zur Softwareentwicklung liefern, bei denen

das Ergebnis ihrer Arbeit durch Offenlegung des Codes und expliziter Erlaubnis der Weitergabe „verschenkt“ wird, d.h. Dritte daraus Nutzen ziehen können. Als Motivationen können genannt werden (vgl. auch Luthiger (2004), Hars und Shaosong (2002), Hertel u. a. (2003), Franck und Jungwirth (2002)):

- Direkter *intrinsischer Nutzen* aus dem Erwerb von Wissen bzw. Humankapital und deren Einsatz im Produktionsprozess, sowie unmittelbarer Nutzen aus der Anerkennung der eigenen Leisung durch die Community (Reputation) (vgl. Kalwey u. a. (2003b), S.57). Problematisch an Erklärungen dieser Art ist aber die schwierige empirische Falsifizierbarkeit, da Nutzenfunktionen beliebige Argumente enthalten können. Dennoch scheinen intrinsische Leistungsanreize eine große Rolle zu spielen (vgl. Osterloh u. a. (2000)).

- Erwerb eines *Reputationssigmals* für den Arbeitsmarkt, um dort die Arbeitsplatzchancen und Entlohnung positiv zu beeinflussen (vgl. Lerner und Tirole (2000), S.14 ff., ?)).

- Der Entwicklungsprozess profitiert auch von den Fehlerreports und Vorschlägen der Nutzer. Diese programmieren zwar nicht selbst, partizipieren aber dennoch am Entwicklungsprozess. Solche freiwilligen Beiträge lassen sich leicht erklären, da hierbei mit geringen Transaktionskosten erreicht werden kann, dass eventuell für den Nutzer spürbare Verbesserungen implementiert werden (vgl. Lerner und Tirole (2000), S.14).

- Selbständige Entwickler können durch die Vermarktung von *Komplementärprodukten* (Schulungen, Überwachung und Konfiguration von OSS-Systemen, Beratung etc.) ein Einkommen erzielen. Die freiwillige Entwicklungsleistung ist insofern eine Investition in ein stabiles und sich dynamisch entwickelndes Marktumfeld, auf dem die Komplementärprodukte aufbauen. Ebenso lassen sich die Beiträge kommerzieller Firmen – häufig Hardwarehersteller – als Teil eines *OSS-Geschäftsmodells* erklären, weil das Engagement Reputation erzeugt und OSS-Produkte sowie damit verbundene Serviceleistungen die Wertschöpfungskette verbessern (z.B. der Absatz von Servern mit vorkonfiguriertem und optimierten Linux-Betriebssystem und entsprechendem Supportvertrag)(vgl. Leiteritz (2004), ?)).

- Schließlich existieren in der OSS-Community gewisse Reziprozitätsvorstellungen, die dazu führen, dass freiwillige Entwickungsleistungen teilweise durch freiwillige Zahlungen bzw. durch *Sponsoring* honoriert werden.

Neben den letztlich mit neoklassischen Rationalitätskonzepten vereinbaren Gründen existieren verhaltenswissenschaftlich orientierte Erklärungsansätze für kooperatives Verhalten, speziell im Bereich öffentlicher Güter, auf die der zuletzt genannte Punkt abzielt. Ein Beispiel ist die Motivation Kooperationsbeiträge zu leisten, weil man die

generellen Ziele und Werte, die „Ethik“ der Community teilt und über den Eigennutz stellt. Die große Bedeutung, welche die theoretische und normative Begründung „freier Software“ in der Community hat (vgl. Stallman (1992), Stallman (2002)), legt nahe, dass solche normgeleiteten und regelgebundenen Verhaltensmuster eine signifikante Rolle spielen. Solches an Normen und Regeln orientierte Verhalten mag aus traditioneller ökonomischer Sicht als „irrational“ oder zumindest „begrenzt rational“ erscheinen. Dennoch existieren Ansätze, solches empirisch robustes Verhalten plausibel zu erklären (vgl. Johnson (2001)). Es ist deshalb davon auszugehen, dass ökonomisches Verhalten nicht ausschließlich durch *Anreize* vermittelt wird.

Über die Frage der Motivation der Leistungserstellung hinaus ist festzuhalten, dass der einzelne Entwickler gar nicht vor der Wahl steht, ob er Eigentums- bzw. Verwertungsrechte an seinem Produkt geltend machen will oder darauf verzichtet. Da fast jede Software hinreichend komplex ist, was eine arbeitsteilige Produktion erfordert, ist der individuelle Entwicklungsbeitrag schwer messbar und eine Aufteilung des Verwertungsanspruchs auf alle Entwickler sehr schwierig. Das gilt insbesondere bei „Kleinstbeiträgen“ sowie für Personen, die durch Testen von Code und Dokumentation zum Entwicklungsprozess beitagen. Die Frage, wem das Softwareprodukt „gehört“, hängt von der Produktionsweise ab: Wird Software im Rahmen eines Prinzipal-Agenten-Systems entwickelt, so hält der Prinzipal die Verwertungsrechte. Wird sie im Rahmen eines Community-Projektes entwickelt, existiert keine Institution, welche Verwertungsansprüche geltend machen kann. Es kann im Gegenteil ein einzelner Entwickler nur dann einen Betrag zum Projekt leisten, wenn er (z.B. durch Anerkennung der GPL) explizit auf solche Ansprüche verzichtet. Somit bleibt dem Entwickler nur die Wahl, innerhalb eines kommerziellen Unternehmens Software zu entwickeln oder sich an einem OSS-Community-Projekt zu beteiligen. In beiden Fällen kann er keinen unmittelbaren Verwertungsanspruch geltend machen[5].

Grundsätzlich besteht auch in der Community die Notwendigkeit, komplexe Softwareprojekte arbeitsteilig, also modular zu bearbeiten und diese Teilprojekte zu koordinieren und zu integrieren. Dies geschieht im Gegensatz zur eher hierarchisch geprägten und durch Verträge konstituierten Organisationsform der Firma auf *dezentralem* Weg (vgl. Ettrich (2004)). Dabei spielen unterschiedliche Akteursgruppen eine Rolle:

- Etablierte freie Entwicklerteams, welche die Kontrolle über den Code des Projektes ausüben und einen kontinuierlichen formalisierten Entwicklungsprozess aufweisen (z.B. dezidierte Release-Politik, Qualitätssicherung, Bug-Report-System). Beispiele: Kernel-Team, KDE-Team, Gnome-Team, XFree86-Team usw.,
- einzelne freie Entwickler und kleinere Teams mit sporadischen Entwicklungsbeiträgen,

[5] Von kleinen Unternehmen, bei denen Unternehmensleitung und Programmierer identisch sind, sei hier abgesehen.

- Softwarenutzer, welche den Entwicklerteams über Fehler (Bugs) berichten oder Wünsche bzw. Verbesserungsvorschläge mitteilen, oder sich an der Dokumentation beteiligen,
- kommerzielle und nicht-kommerzielle OSS-Distributoren, die z.T. eigene Entwicklungsbeiträge leisten (SuSE, RedHat, Mandrake bzw. Community-Projekte wie Debian, Slackware, Fedora),
- kommerzielle Anbieter mit einem OSS-Geschäftsmodell und mit eigenen OSS-Entwicklungsbeiträgen (Sun, IBM, HP, Novell usw.),
- vernetzende und Interessen bündelnde Institutionen wie die Free Software Foundation (FSF) und die Open Source Development Labs (OSDL).

Die an einem Projekt beteiligten Entwickler kommunizieren dabei weitgehend – und häufig ausschließlich – über das Internet. Räumliche Entfernungen spielen ebenso wenig eine Rolle wie Persönlichkeitsmerkmale (z.B. Alter, Geschlecht, sozialer Status). Für jedes Softwareprojekt gibt es zwar klare Verantwortlichkeiten der Projektteams und ein transparentes System, nach welchem Entwicklungsbeiträge eingereicht, bewertet und in das Projekt eingepflegt werden (z.B. das „Concurrent Versions System“ CVS). Dieser Prozess beinhaltet die Prüfung der Qualität, Konsistenz und Fehlerfreiheit des Codes, die zwar nicht perfekt, aber mit den entsprechenden Prozessen kommerzieller CSS-Entwicklung vergleichbar sind. Jedoch können Projektverantwortliche lediglich Teilbeiträge koordinieren, da es keine zentrale Instanz gibt, welche bestimmte Lösungen, Entwicklungsrichtungen, Standards usw. vorschreibt. Dieser Umstand eines fehlenden Prinzipals führt zunächst zu der Gefahr, dass

- Teilprojekte nicht weiter bearbeitet werden („Verwaisung“), da sich keine freiwilligen Entwickler dafür finden,
- die Entwicklungen sich zu sehr aufspalten, so dass sich die Ressourcen auf konkurrierende und nur teilweise kompatible Lösungen verteilen (Forking),
- die Kontinuität der Weiterentwicklung nicht unbedingt sichergestellt werden kann.

Diese bei einer rein dezentralen Koordination auftretenden Probleme werden von Kritikern des OSS-Modells als Beleg für dessen geringe Effizienz und Leistungsfähigkeit gewertet. Solche Koordinationsprobleme, die bei hinreichend „großen“ und im Markt etablierten Softwareprojekten kaum auftreten, sind jedoch der Preis für einige Vorzüge, die das Modell der dezentralen Produktion aufweist: Entscheidendes Merkmal ist, dass in jedem Stadium des Entwicklungsprozesses der Source-Code öffentlich zugänglich ist. Dies eröffnet erst die Möglichkeit, dass weltweit verstreutes Humankapital mobilisiert und für die Entwicklung genutzt werden kann, was sich vor allem bei der Durchleuchtung des Codes auf mögliche Fehler und Sicherheitslücken zeigt (vgl. Gehring (2004)):

Die Tatsache dass eine große Anzahl Nutzer den Source-Code begutachtet, sich über mögliche Fehler austauscht sowie Verbesserungsvorschläge diskutiert und Lösungen entwickelt, wird daher auch häufig als der große Vorteil der OSS genannt.

Die Tendenz, dass sich gerade wegen offener Standards (d.h. dem leichten „Markteintritt“ neuer Projekte) parallel unterschiedliche Lösungen entwickeln, Softwareprojekte „zersplittern“ mit dem Risiko, dass Teilprojekte nicht mehr weiter verfolgt werden, kann sich aber auch positiv für den Nutzer auswirken: Der Nutzer ist nicht auf eine bestimmte Softwarelösung festgelegt, sondern kann frei aus einer Vielfalt unterschiedlicher Lösungen wählen. Diese *Wahlfreiheit* betrifft nicht nur die Frage, z.B. welches Textverarbeitungsprogramm oder Browser er benutzt, sondern auch die freie Wahl der Desktop-Oberfläche, der Systemsteuerung (Kontrolle über das Betriebssystem, die Hardware-Ressourcen und die instalierte Software) und sogar die Konfiguration der Fähigkeiten des Betriebssystem-Kernels. Diese Wahlfreiheit wird vor allem dadurch ermöglicht, dass die Open-Source-Idee auch *offene Standards* impliziert[6]. Auf diese Weise ist es sehr viel leichter, die Daten- und Schnittstellenkompatibilität konkurrierender Softwarelösungen sicherzustellen (vgl. Kuhlmann (2004)). Dies senkt erheblich die switching costs zwischen unterschiedlichen OSS-Produkten. Da die Produkte selbst kostenlos sind, kann sich der Nutzer sehr viel einfacher mit unterschiedlichen Lösungen und Varianten vertraut machen (Erfahrungsgut-Eigenschaft), bevor er sich entscheidet. Ein weiterer Vorzug ist, dass der Kunde zumindest die Kontrolle über den Code und somit Transparenz darüber hat, was auf seinem Computersystem geschieht. Dies ermöglicht zumindest im Prinzip die Anpassung des Codes an die eigenen Bedürfnisse.

Firmenkunden sind auf Kontinuität und Support angewiesen. Dies zieht eine Professionalisierung der OSS-Community nach sich. So werden Firmenkunden kaum eigenständig OSS-Produkte zusammenstellen und pflegen, sondern auf bereits fertig vorkonfigurierte Distributionen von Anbietern wie z.B. RedHat, SuSE, Sun oder auch IBM und HP (im Serverbereich) zurückgreifen, welche ihre Produkte kontinuierlich dem technischen Stand anpassen, die Kompenenten aufeinander abstimmen und auch den gewünschten Support leisten. Hier besteht kaum noch ein Unterschied zu kommerziellen Softwarelösungen. Auch für OSS gibt es eine wachsende Zahl von IT-Dienstleistern, welche solche Komplementärprodukte und -leistungen anbieten.

## 4 Effizienzwirkungen und Wechselbeziehungen von OSS und CSS

Die beiden Formen der Produktion und Bereitstellung von Software unterscheiden sich in ihren statischen und dynamischen Effizienzwirkungen, beeinflussen sich aber auch gegenseitig. Einige der folgenden Überlegungen unterstreichen, dass trotz fehlender unmittelbarer Steuerung durch den Markt und entsprechender Probleme dezentraler Koordination das OSS-Modell einige Vorzüge aufweist. Man kann erwarten, dass eine

[6]Die Umkehrung gilt jedoch nicht.

Mischung kommerzieller und freier Software volkswirtschaftlich effizient ist (vgl. Shapiro und Varian (2003)). Für eine volkswirtschaftliche Beurteilung der Effizienzwirkungen sind sehr unterschiedliche Aspekte zu berücksichtigen. Dazu zählen nicht nur die unmittelbaren Wirkungen auf dem Softwaremarkt, sondern auch die Frage einer effizienten Verwendung des Produktionsfaktors „Humankapital“, die Rolle positiver Spillover des Wissenskapitalstocks, der Orientierung an den Präferenzen der Kunden und dynamische Wirkungen auf Wettbewerb und den Innovationsprozess.

### *4.1 Direkte Effizienzwirkungen auf dem Softwaremarkt*

Bei einer partialanalytischen Betrachtung des Softwaremarktes spielen die diskutierten Eigenschaften von Software eine Rolle. Da zunächst keine Rivalität im Konsum herrscht und Software zu Grenzkosten von (nahezu) Null reproduziert werden kann, ist ein Marktpreis von Null optimal, um eine effiziente Allokation zu erreichen. Das aber führt zu dem Problem, dass kein Anreiz besteht, Software zu entwickeln, obwohl eine positive Zahlungsbereitschaft vorhanden ist, so dass die „Effizienz ex ante“ nicht gewährleistet wäre. Das kommerzielle CSS-Modell löst das Problem durch kostenpflichtige Lizenzen zur Nutzung des binären Kompilats der Software. Der gewinnmaximale Marktpreis für Softwarelizenzen muss über den Produktlebenszyklus hinweg eine Deckung der Entwicklungskosten garantieren. Auf diese Weise ist das Anreizproblem behoben, jedoch um den Preis einer deutlich verringerten Konsumentenrente, d.h. stark verringerter „Effizienz ex post“. Dies unterstreicht das Ergebnis von Quah (2003), dass bei digitalen Gütern die Definition von Verfügungsrechten (intellectual property rights) weder eine notwendige noch hinreichende Bedingung für deren effiziente Bereitstellung ist.

Das OSS-Modell verzichtet auf das Ausschlussprinzip und stellt Software kostenlos zur Verfügung, was der wohlfahrtsökonomischen Bedingung einer „first best“-Allokation bei nichtrivalisierendem Konsum entspricht. Der Anreiz zur Produktion des Gutes muss folglich auf anderem Weg sichergestellt werden. Dazu wurden eine Reihe von Gründen diskutiert. Die Entwicklungskosten (Faktoreinsatz) können somit nicht über den Marktpreis für Software gedeckt werden. Die MICE-Studie argumentiert, dass dadurch eine Verzerrung des Preissystems induziert wird, die in einer Marktwirtschaft Fehlallokationen nach sich ziehe (Kalwey u. a. (2003b)). Das Argument ist aus mehrren Gründen zu kritisieren:

- Das Argument der Studie basiert auf der Vorstellung eines walrasianischen Preissystems, welches auf vollständig kompetitiven Märkten eine effiziente Allokation sicherstellt. Nun liegen aber gerade durch die ausführlich diskutierten Eigenschaften des Gutes Software gleich mehrere Gründe für Marktversagen bzw. unvollkommenen Wettbewerb vor. Bei Grenzkosten von nahezu Null besteht der Marktpreis aus einem Deckungsbeitrag für die Entwicklungskosten und einer Monopolrente. Zu bedenken sind außerdem nachfrageseitige Wechselkosten, so dass

die dadurch verringerte Preiselastizität ebenfalls vom Anbieter als Preissetzungsspielraum ausgenutzt werden kann. Von einem Preissystem, welches man sich im Referenzmodell kompetitiver Märkte für rein private Güter vorstellen kann, ist hier schon aufgrund der Eigenschaften des Gutes Software und (deshalb) der besonderen Angebots- und Nachfragebedingungen nicht auszugehen.

- Ein Teil der OSS-Entwicklungsleistung wird aus intrinsischen Gründen freiwillig geleistet. Hier besteht eine Schwierigkeit, weil nicht-kommerzielle Tätigkeiten in der ökonomischen Theorie als „Freizeit" behandelt werden, in der per Definition gerade kein Faktoreinsatz geleistet wird. Ein nicht unerheblicher Teil der Opportunitätskosten der OSS-Entwicklung liegen in entgangener Freizeit, aber nicht im Entzug von Ressourcen aus anderen produktiven Verwendungsmöglichkeiten. Bei gegebener Lösung des Arbeits-Freizeit-Kalküls eines Programmierers bestehen dessen Opportunitätskosten aus den Freizeitaktivitäten, auf die er verzichtet. Diese würden aber ebenfalls nicht vergütet werden.

- Ein Teil der OSS wird im Rahmen von OSS-Geschäftsmodellen entwickelt. Hier müssen, wie die MICE-Studie korrekt argumentiert, die Preise der Komplementärprodukte die OSS-Entwicklungskosten mit abdecken. Allerdings stellt der Umstand, dass der Kunde ein *Bündel* von Leistungen nachfragt, weil nur das Bündel z.B. aus Software, Schulung, Installation und Support die gewünschte Problemlösung darstellt, für die er eine Zahlungsbereitschaft formuliert, keine Besonderheit von OSS dar. Man kann argumentieren, dass in OSS-Geschäftsmodellen Leistungsbündel produziert und zu marktgerechten Preisen verkauft werden, wobei zugelassen wird, dass die Teilleistung „Software" auch von Dritten, die keine Zahlungsbereitschaft für das gesamte Leistungsbündel haben, kostenlos mitgenutzt werden kann.

- Auch kommerzielle Anbieter verkaufen Bündel von Leistungen. Dabei ist es aus Sicht des Anbieters strategisch rational, den Wettbewerb dadurch zu behindern, dass die dominante Position in einem Teilmarkt dazu genutzt wird, das dominierende Produkt nur im Bündel mit einer anderen Software eines anderen Teilmarktes zu vertreiben, um auf dem zweiten Teilmarkt ebenfalls eine dominierende Position zu erhalten und Wettbewerber zu verdrängen („leveraging", Beispiel: Microsoft Windows im Verbund mit dem Internet-Explorer oder dem Media-Player). Auch dies kann als „Quersubventionierung" und insofern als Verzerrung des Preisgefüges gewertet werden, für die sogar ein strategischer Anreiz im Marktsystem besteht.

Unter dem Gesichtspunkt der statischen Effizienz kann angesichts des Öffentlichen-Gut-Charakters von OSS ein Marktpreis von Null wohl kaum kritisiert werden. Hinzu kommt, dass OSS im Gegensatz zu CSS einen Beitrag zum allgemein verfügbaren Wissenskapitalstock beiträgt, also positive externe Effekte hat, die bei einer statischen Effizienzanalyse berücksichtigt werden müssen. Darauf wird im nächsten Abschnitt näher

eingegangen. Aufgrund des notwendigerweise unvollkommenen Wettbewerbs auf kommerziellen Softwaremärkten ist es nicht legitim, dem OSS-Modell Preisverzerrungen und Effizienzdefizite vor dem Hintergrund eines vollkommenen walrasianischen Gleichgewichtsmodells vorzuwerfen und so einen Nachteil gegenüber CSS zu begründen.

Im Hinblick auf die *dynamischen Effizienzwirkungen* spielen permanente Anpassungen an sich verändernde Kundenwünsche und die Innovationstätigkeit eine Rolle. Da die Orientierung an Kundenpräferenzen gesondert behandelt wird, werden hier nur einige Überlegungen zum Innovationsanreiz dargelegt. Kommerzielle Softwareanbieter haben einen Innovationsanreiz aus drei Gründen:

- Falls es konkurrierende Hersteller geben sollte, liegt der Innovationsanreiz (Erlangung von Wettbewerbsvorteilen) auf der Hand. Bei stark differenzierten Produkten und einer starken Monopolisierung aufgrund von Netzwerk-Externalitäten ist dieser Anreiz eher moderat, denn ein Innovationsvorsprung zahlt sich nur dann aus, wenn er Nachfrager aus einem konkurrierenden Netzwerk zu einem Wechsel bewegen, also dessen switching costs überwinden kann.

- Ein entscheidender Innovationsanreiz besteht darin, dass Software keinem Verschleiß unterliegt und somit die Zahl der verkauften Lizenzen schnell zu einer Sättigung führen würde. Durch permanente Innovation wird eine anhaltende Nachfrage nach Software bzw. Updates erzeugt. Eine Monopolrente kann nur solange abgeschöpft werden, wie die Nachfrage an Lizenzen noch nicht gesättigt ist.

- Zwar veraltet nicht die einzelne Softwarelizenz, jedoch das technische Wissen, welches in der Software steckt. Regelmäßige oder auch fallweise Innovationen erfordern hohe Entwicklungskosten. Diese stellen dann *sunk costs* dar, welche eine Markteintrittsbarriere für potentielle Konkurrenten darstellen (Baumol u. a. (1982)). Innovationen sind also auch als strategische Markteintrittsbarriere notwendig, weil sonst die Monopolstellung durch potenzielle Innovatoren bedroht wird, die in den Markt eindringen könnten.

Bei der OSS-Entwicklung scheinen diese Gründe zunächst nicht zu greifen. Die Motivation für *innovative* Lösungen entsprechen im Kern denselben Motivationen, weshalb überhaupt freiwillige Entwicklungsbeiträge geleistet werden. Wegen der fehlenden unmittelbaren kommerziellen Verwertung der Software ließe sich vermuten, dass daher auch der Innovationsanreiz schwächer ausgeprägt ist, zumal stets die Gefahr einer Parallelentwicklung innovativer Ideen besteht. Auf der anderen Seite bietet gerade der fehlende Verwertungsdruck auch die Möglichkeit, statt inkrementeller Verbesserungen auch das höhere Risiko völlig neuartiger Konzepte und Problemlösungen einzugehen, deren Markterfolg sehr ungewiss ist (vgl. Kogut und Matiu (2001)). Auf Grund der zunehmenden Professionalisierung des OSS-Bereichs können, da immer mehr Entwicklungsbeiträge vor dem Hintergrund eines OSS-Geschäftsmodells getätigt werden, im

Prinzip auch dieselben Argumente geltend gemacht werden, wie sie für kommerzielle Softwareanbieter angesprochen wurden. Eine abschließende Bewertung, welches Modell auf lange Sicht die höhere Innovationsleistung (und somit dynamische Effzienz) hervorbringt, kann nicht gegeben werden.

### *4.2 Spillovereffekte und Wissensverbreitung*

Wird Software aufgrund durchsetzbarer Verfügungsrechte zum Club-Gut, dann wird zwar einerseits das Trittbrettfahrerproblem gelöst, auf der anderen Seite aber auch erhebliche positive Spillover-Effekte des codierten Wissens ausgeschlossen. Da bei CSS der Source Code geheimgehalten wird, kann kein Dritter von den Problemlösungstechniken lernen, diese ggf. verbessern, Sicherheitslücken aufspüren, Algorithmen für ähnliche Problemstellungen übernehmen etc. Die Verbreitung dieses Wissens wäre jedoch für den allgemeinen technischen Fortschritt von großer Bedeutung. Aus der Neuen Wachstumstheorie ist bekannt, dass ökonomische Aktivitäten, welche zu positiven Externalitäten in Gestalt von frei verfügbarem Wissen führen, positiv für das Wachstum sind und ein zu geringes (pareto-ineffizientes) Niveau solcher Aktivitäten zu verzeichnen ist, wenn diese allein durch Marktpreissignale gesteuert werden (vgl. z.B. Barro und Sala-i Martin (1995), S.146 ff.). Dieser Verlust an positiven Wissens-Spillovern lässt sich mit dem Hinweis darauf rechtfertigen, dass es ohne das Ausschlussprinzip via Lizenzen gar nicht erst zur Entwicklung dieses Wissens käme. Allerdings zeigt die *Existenz* von OSS, dass dies ein kontrafaktisches Argument ist.

Auch ein Teil des in CSS enthaltenen Wissens ist öffentlich zugänglich. Dieses bezieht sich allerdings nur auf allgemeine Konzepte (z.B. Bedienung mit Fenstern und Pull-Down-Menüs, Drag-and-Drop-Technik, Plug-In-Technik usw.), nicht auf detallierte programmiertechnische Lösungen. Außerdem sind im Softwaredesign – wie in jedem anderen Produkt – Informationen über die Kundenpräferenzen enthalten, wie sie von den Firmen wahrgenommen werden. Von diesen sehr allgemeinen Wissensinhalten profitieren sowohl kommerzielle Konkurrenten als auch die OSS-Community. Allerdings können zumindest im US-amerikanischen Rechtssystem auch solche Spillovereffekte von konzeptionellem Wissen durch Softwarepatente verhindert werden.

Beim Open-Source-Konzept ist das erzeugte öffentlich zugängliche Wissen sehr viel umfangreicher, und die OSS-Entwicklung beruht wesentlich auf der Ausnutzung der dadurch induzierten Spillovereffekte. Zwar wird in der Community die freie Verfügbarkeit des Codes häufig durch idealistische und normativ-ethische Argumente begründet (vgl. Stallman (2002)), jedoch ist schon aus ökonomischer Sicht der Beitrag zu einem öffentlichen Wissenskapitalstock wegen der Externalitäten und damit produktivitätserhöhenden Wirkung positiv zu werten.

Das frei verfügbare Wissen ist jedoch nur im Rahmen der OSS-Lizenzpolitik „frei“. Die GPL etwa untersagt die Verwendung des Open-Source-Codes für kommerzielle

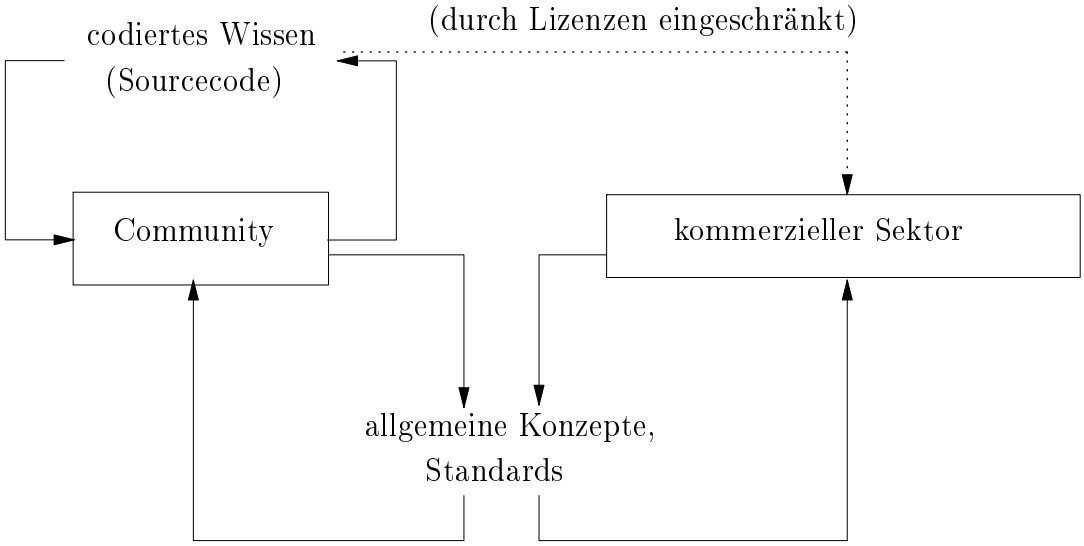


Abbildung 1: Wissens-Spillovereffekte

Software. Das akkumulierte Wissenskapital soll der Community zugute kommen, weil es für das Funktionieren des OSS-Entwicklungsprozesses unentbehrlich ist, ein Trittbrettfahren kommerzieller Anbieter soll aber verhindert werden. Durch diese moderate Form des Ausschlussprinzips werden positive Spillovereffekte eingeschränkt, und zwar dann, wenn Individuen beispielsweise darauf verzichten OSS-Elemente zu verwenden, da sie das Ergebnis nicht kommerziell verwerten können[7]. Diese restriktive Eigenschaft der GPL verringert zwar einerseits die Spillovereffekte, vermeidet aber dadurch einen Wettbewerbsnachteil gegenüber CSS, so dass die Existenz der Institution einer OSS-Community gesichert wird. Es ist leicht einsehbar, dass die Nachprüfbarkeit dieses kommerziellen Verwertungsverbotes sehr schwierig ist. Jedoch ist es in Einzelfällen möglich gewesen, Verstöße gegen die GPL juristisch zu ahnden (Heise News (2004e)). Der kommerzielle Softwaresektor kann nur von allgemeinen OSS-Konzepten (einschließlich der dort verwendeten Standards und Schnittstellen) und dem programmiertechnischen „Lösungsstil“ des Source-Codes lernen, jedoch kann der Code nicht direkt übernehmen werden. Eine schematische Übersicht über Spillovereffkte innerhalb der OSS-Community sowie zwischen OSS und CSS bietet Abbildung 1.

[7]Anzumerken ist, dass einige Firmen mit OSS-Geschäftsmodell dazu übergehen, ihre Produkte nach einem „Dual License“ Modell anzubieten, also entweder kostenlos gemäß der GPL, oder kostenpflichtig nach einer weniger restriktiven Lizenz, welche eine kommerzielle Verwendung des Codes erlaubt. Beispiele: MySQL, Trolltech.

### *4.3 Zur Humankapitalallokation*

Da der Open-Source-Code nicht am Markt verkauft und somit dessen Entwicklung nicht unmittelbar durch den Markt gesteuert wird, argumentiert die MICE-Studie, dass hier volkswirtschaftliche Ressourcen, insbesondere Humankapital, nicht effizient verwendet werden und es somit zur Fehlallokation von Ressourcen komme (vgl. Kalwey u. a. (2003b), S.70 ff.). Die Teile des Humankapitals, die für OSS-Entwicklung eingesetzt werden, würden dem kommerziellen Sektor entzogen und würden daher nicht oder kaum zur Wertschöpfung beitragen. Außerdem spiegele der Marktpreis von Null für OSS nicht den erforderlichen Ressourceneinsatz wider, so dass aufgrund eines verzerrten Preissystems der Humankapitaleinsatz fehlgesteuert werde.

Ein wichtiger Punkt bei der Frage, ob eine in OSS eingesetzte Humankapitaleinheit weniger produktiv ist, als wenn sie für kommerzielle Softwareentwicklung eingesetzt würde, ist, dass Lizenzumsätze unmittelbar die Wertschöpfung repräsentieren, während beim kostenlosen Download von OSS keine Markttransaktion stattfindet und die Statistik folglich keine „Wertschöpfung“ ausweist, obwohl das Produkt genauso hochwertig sein kann wie kommerzielle Software. Die Wertschöpfungswirkung von OSS wird somit zwangsläufig statistisch unterschätzt. Das ist ein Problem der Statistik, nicht der OSS. Es besteht a priori kein Grund für die Annahme, dass aus rein *technischer* Sicht die Durchschnitts- oder Grenzproduktivität des Humankapitaleinsatzes im OSS-Sektor geringer ist.

Ein zweiter wichtiger Punkt betrifft die Frage, ob man eine Ineffizienz aus dem *Entzug* von Humankapital aus dem kommerziellen Sektor ableiten kann. Das Argument lässt sich auch so formulieren, dass im OSS-Bereich eingesetztes Humankapital (zu) hohe Opportunitätskosten verursache. Dabei wird erstens die Frage ausgeblendet, woher das Humankapital stammt, und zweitens wird unterstellt, dass eine solche Wahlmöglichkeit, wo das Humankapital eingesetzt wird, überhaupt besteht. Im Folgenden wird argumentiert, dass zum einen die Community selbst Humankapital erzeugt und somit das Produktionspotential der Volkswirtschaft erhöht. Zum anderen wird dargelegt, dass aus unterschiedlichen Gründen die Softwareentwickler, welche Träger des Humankapitals sind, sich gar nicht vor die Entscheidung gestellt sehen, ob sie im OSS- oder CSS-Sektor Software entwickeln wollen. Wenn aber keine Alternativen zur Wahl stehen, ist auch die unterstellte Opportunitätskostenüberlegung hinfällig.

Tabelle 1 zeigt neben den Möglichkeiten, Humankapital für die CSS- oder OSS-Entwicklung einzusetzen, auch zwei unterschiedliche Herkunftsarten von Humankapital auf. Humankapital kann zum einen im „kommerziellen Sektor“ erworben werden. Damit ist gemeint, dass entweder durch staatliche oder privatwirtschaftliche Bildungsmaßnahmen Humankapital aufgebaut wird (also durch Ressourceneinsatz), oder durch learning-by-doing (also als Spillovereffekt produktiver Tätigkeit) entsteht. Dabei hat jedes Individuum als Träger von Humankapital bereits in einem Arbeits-Freizeit-Kalkül

| | | Einsatz HK | |
|---|---|---|---|
| | | CSS | OSS |
| Herkunft HK | Freizeit-tätigkeit | A | B |
| | kommerzielle Tätigkeit | C | D |

Tabelle 1: Herkunft und Verwendung von Humankapital

entschieden, wieviel Zeit es im kommerziellen Sektor einsetzen will. In der mikroökonomischen Theorie wird Freizeit als nicht-produktive Tätigkeit behandelt und Freizeit-Aktivitäten haben demnach nur konsumtiven Charakter. Diese sehr stilisierte Sicht kann eine Reihe ökonomischer Phänomene nicht abbilden. Im vorliegenden Fall ist festzustellen, dass im Rahmen von Freizeit- oder zumindest nicht primär erwerbswirtschaftlichen Aktivitäten

- Humankapital akkumuliert wird (Erlernen von Programmiersprachen, Erwerb von IT-Wissen) und
- Humankapital produktiv in der OSS-Entwicklung eingesetzt wird.

Somit kann man argumentieren, dass der Teil des Humankapitals, welcher in der Freizeit produktiv eingesetzt wird, nicht dem kommerziellen Sektor verloren geht. Der Entwickler steht allenfalls vor der Opportunitätsentscheidung, einen OSS-Beitrag zu leisten oder aber seine Freizeit anderweitig zu gestalten. Dieses Argument lässt sich je nach Herkunft des Humankapitals differenzieren:

- In der Freizeit akkumuliertes und in der Freizeit für OSS eingesetztes Humankapital entzieht dem kommerziellen Sektor keine Ressourcen, sondern trägt sowohl zur Vergrößerung des Produktionspotentials als auch zur Erhöhung des Outputs bei, wobei letzterer allerdings nicht durch Marktpreise bewertet und insofern statistisch unteschätzt wird. Dies betrifft die empirisch schwer zu ermittelnde Zahl autodidaktischer Freizeitprogrammierer, aber auch der Softwarenutzer, die durch den Umgang mit Software Humankapital aufgebaut haben (Erfahrungsgut-Eigenschaft) und am OSS-Entwicklungsprozess partizipieren.
- Im kommerziellen Sektor akkumuliertes und *zusätzlich* zur Erwerbsarbeit auch in der Freizeit eingesetztes Humankapital wird ebenfalls nicht dem kommerziellen Sektor entzogen, sondern lediglich besser ausgelastet. Dies betrifft z.B. ausgebildete Programmierer bzw. Informatiker, die sich neben Studium oder Berufstätigkeit an der OSS-Entwicklung beteiligen.

- Wird in der Frezeit Humankapital aufgebaut und OSS-Beträge geleistet mit dem Ziel, ein Reputationssignal für den Arbeitsmarkt zu erwerben, dann profitiert der kommerzielle Sektor von der Akkumulationsleistung der Community, ohne selbst Ressourcen für die Humankapitalbildung aufbringen zu müssen.

Das Argument, dass OSS dem kommerziellen Sektor Humankapitalressourcen entziehe und weniger produktiven Verwendungen zuführe, betrifft allein den Fall, dass eine Wahlentscheidung „C *oder* D" zugunsten von D getroffen wird (vgl. Tabelle 1). Angesichts fehlenden empirischen Materials kann die statistische Bedeutung dieser Konstellation momentan nicht beurteilt werden. Da bei D das Humankapital nicht unmittelbar nach seiner Grenzproduktivität entlohnt wird, muss es entweder komplexere Opportunitätsentscheidungen gegeben haben (z.B. OSS-Geschäftsmodell versus Prinzipal-Agenten-Vertrag), oder es haben andere Motivationen als rein erwerbswirtschaftliche eine Rolle gespielt. Obwohl sich die Wertschätzung für das Produkt nicht in einem Marktpreis niederschlägt, kann nicht davon ausgegangen werden, dass im Fall von D das Humankapital technisch ineffizient verwendet wird.

Eine Alternativentscheidung, ob das Humankapital – gleich welcher Herkunft – für OSS oder CSS eingesetzt werden soll, besteht nicht immer. Eine Eigenschaft der OSS-Community ist es gerade, auch geringe Entwicklungsbeiträge von solchen Personen zu integrieren, welche nicht über genügend Humankapital verfügen, um überhaupt im kommerziellen Sektor eingesetzt zu werden. Wenn beispielsweise ein förmlicher Abschluss als Diplom-Informatiker fehlt, dann fehlt das *Signal*, welches die asymmetrische Informationsverteilung zwischen dem Programmierer und dem potentiellen Arbeitgeber (bezüglich der Programmier-Kompetenz des Programmierers) abbauen kann. Dieses kann eine Beschäftigung im kommerziellen Sektor behindern oder sogar unmöglich machen. Der Erwerb entsprechender Reputation durch hochwertige OSS-Entwicklungsbeiträge kann jedoch ein äquivalentes Signal darstellen. Darüber hinaus besteht die Alternativentscheidung auch dann nicht, wenn OSS-Entwickler bereits erwerbswirtschaftlich tätig sind und ihre OSS-Beiträge freiwillig leisten.

### *4.4 Präferenzsteuerung versus Happy-Engineering*

Ein Argument für die mangelnde Effizienz von OSS besteht darin, dass sich kommerzielle Software am Markt behaupten muss, so dass der Wettbewerb für eine Steuerung der Softwareentwicklung nach den Präferenzen der Anwender erfolge. Dagegen richte sich die Entwicklung von OSS vor allem nach den Bedürfnissen der Entwickler selbst, das sogenannte „Happy-Engineering" (vgl. Kalwey u. a. (2003b), S.66 ff.). Diesem Argument stehen jedoch einige Einschränkungen und Gegeneffekte gegenüber. Außerdem sollte nicht ignoriert werden, dass OSS-Produkte in der Einschätzung der Anwender in einigen Bereichen qualitativ kommerziellen Produkten gleichwertig sind, in Einzelfällen sogar überlegen sind (z.B. Serverprodukte). In anderen Bereichen sind allerdings auch

Defizite zu verzeichnen, die aber allmählich abgebaut werden (Heise News (2004h), Heise News (2004f)). So sind die Leistungsunterschiede z.B. zwischen MS Office und OpenOffice nicht so signifikant groß, dass sie zu einer entsprechenden hohen Zahlungsbereitschaft für MS Office führen müssen. Die tatsächlich vorhandene Zahlungsbereitschaft spiegelt zu einem großen Teil die Opportunitätskosten des Wechsels wider. Im Serverbereich ist der Marktanteil OSS-basierter Produkte sogar erheblich, was als Indikator für eine Deckung mit den Kundenbedürfnissen ist (vgl. Heise News (2004a)). Nach dem Happy-Engineering-Argument ist dieser Befund kaum zu erklären.

In einem kompetitiven Markt beruht das Argument der Präferenzsteuerung darauf, dass der Kunde jederzeit auf ein Konkurenzprodukt ausweichen kann, bei dem ihm das Verhältnis von Zahlungsbereitschaft und Nutzen günstiger erscheint. Dadurch haben Anbieter einen Anreiz, Kundenpräferenzen aufzuspüren und ihnen zu entsprechen. Dieser Anreizmechanismus wird aufgrund der Eigenschaften von Software und der dadurch bedingten Marktunvollkommenheiten eingeschränkt:

- Wegen der Erfahrungsgut-Eigenschaft kann der Kunde *vor* dem Kauf einer Lizenz nur bedingt feststellen, in wie weit das Produkt seinen Wünschen entspricht. Vermutet er (nach dem Kauf) bei einem Konkurrenzprodukt, dass dieses seinen Präferenzen eher entspricht, hat er allein durch die Lizenzzahlung und die akkumulierten software-spezifischen Erfahrungen Wechselkosten. Aus dem Nicht-Wechsel zum Konkurrenzprodukt kann somit nicht geschlossen werden, dass das aktuelle Produkt den Präferenzen besser entspricht.
- Wegen der Netzwerk-Externalitäten werden sich Kunden auch dann für ein bestimmtes Gut entscheiden, wenn sie wissen, dass es zwar ihren Präferenzen weniger gut entspricht, aber die installlierte Basis an Softwareeinheiten hinreichend groß ist. Der Lock-In-Effekt verhindert zum einen den Wechsel und bestimmt zum anderen die Erst-Entscheidung für oder gegen ein bestimmtes Softwaresystem.
- Auf der Anbieterseite führen sowohl die Netzwerk-Effekte als auch die spezielle Kostenstruktur zu engen Oligopolen oder gar monopolistischen Marktstrukturen. Dadurch wird schlicht die Wahlmöglichkeit der Nutzer stark beschränkt.

Die Präferenzsteuerung im kommerziellen Softwaremarkt solte nicht so beurteilt werden, als würde es sich um einen vollkommenen kompetitiven Markt handeln. Ein kommerzieller Anbieter darf sich lediglich nicht so weit von den Kundenwünschen entfernen, dass der Vorsprung eines konkurrierenden Systems die Wechselkosten überkompensiert. Ein Teil der Innovationsleistung kommerzieller Hersteller ist sicherlich auf dieses Ziel der Sicherung der Marktposition zurückzuführen. Eine unmittelbare strategische Folge der Marktstruktur ist es aber auch, die Marktmacht dazu zu nutzen, sowohl potenzielle Konkurrenten vom Markt fernzuhalten, als auch den Kunden einen Wechsel zu erschweren, indem deren Wechselkosten strategisch erhöht werden. Dazu dienen u.a.

auch proprietäre Standards und Datenformate statt offener Standards. Ein aktuelles Beispiel ist das Bestreben Microsofts, in den Markt für mobile Audioplayer einzudringen, in welchem bisher der iPod von Apple dominiert, der auf dem verbreiteten offenen Standard MP3 basiert. Microsoft will dagegen das eigene WMA-Format durchsetzen. Gelingt dies, so ist – anders als bei Apples Audioplayer – der Marktzutritt Dritter praktisch unmöglich, weil die WMA-Spezifikation Dritten nicht zugänglich gemacht wird. Was wie eine Innovationsleistung im Sinne der Kundenpräferenzen aussieht, wirkt auf der anderen Seite als Wettbewerbsbeschränkung. Diese schränkt auch die Wahlfreiheit der Konsumenten ein, weil es bei einem proprietären Standard nur einen Anbieter gibt, während bei einem offenen Standard zwar ebenfalls Netzwerkexternalitäten existieren, die aber den Marktzutritt konkurrierender und (den Kundenpräferenzen entsprechend) differenzierter Produkte nicht behindern.

Bei der Entwicklung von OSS kommt es zum einen auf die Entwicklermotivation an, zum anderen unterscheiden sich auch die Mechanismen der Präferenzermittlung von denen kommerzieller Softwareproduktion. Programmierer, die sich aus intrinsischer Motivation oder aus Reputationsgründen an OSS-Projekten beteiligen, haben keinen unmittelbaren ökonomischen Anreiz, sich an den Anwenderwünschen zu orientieren. Sie werden vielmehr ihre Leistung als technische Herausforderung begreifen. Auf der anderen Seite bedeutet aber gerade die Feststellung, dass es außer ökonomischen auch andere Anreize für Entwickler gibt, Software zu produzieren, dass auch alternative Anreize existieren können, welche zu einer Kundenorientierung führen. So kann der Umstand, dass ein OSS-Produkt aufgrund seiner hohen Qualität bzw. Kundenorientierung einen hohen Marktanteil hat, den intrinsischen Entwickleranreiz ansprechen, und ein hohes Reputationssignal darstellen, wenn die Mitarbeit an einem populären Softwareprodukt eine entsprechende Reputation verschafft.

Die Präferenzermittlung der Kunden erfolgt im OSS-Bereich sehr stark auf direktem Weg über das Internet: Dieses ist die Kommunikationsplattform für die Produktentwicklung und kann von jedem Anwender zu vernachlässigbaren Transaktionskosten genutzt werden. Aufgrund des unmittelbar fehlenden Drucks, diesen geäußerten Wünschen nachzukommen und der weniger stark ausgeprägten Möglichkeit, Entwicklerressourcen gezielt und massiv auf bestimmte Kundenprobleme zu orientieren, sind gewisse Defizite bei der Präferenzorientierung durchaus möglich. Vereinzelte Studien belegen zwar einerseits gewisse Rückstände, konstatieren aber auch deutliche Fortschritte in Richtung Nutzerorientierung (Heise News (2004f)). Qualitätssicherung und Benutzerfreundlichkeit gehören zumindest bei großen OSS-Projekten zu den Punkten, deren Berücksichtigung sich auch institutionell widerspiegelt (z.B. usability.kde.org). Die institutionelle Verankerung der Kundenorientierung im Projekt kann durchaus eine Selbstbindungswirkung entfalten.

Ein wichtiger Aspekt der Professionalisierung und Kundenorientierung von OSS stellt aber auch die Tatsache dar, dass immer mehr Firmen jeglicher Größenordnung OSS-Geschäftsmodelle entwickeln. Die Vermarktung von Komplementärprodukten ist längst

kein Nischengeschäft mehr, so dass ein kommerzielles Interesse an einer großen installierten Basis an OSS-Produkten, d.h. an einer entsprechenden Netzwerkgröße vorhanden ist. Dies lässt sich vor allem durch überzeugte Kunden erreichen. Auch wenn hier die Kundenorientierung lediglich auf dem Umweg der Komplementärgutvermarktung erfolgt, so muss dieser Mechanismus nicht unbedingt weniger effektiv sein als bei kommerzieller Softwareentwicklung, wo die Konsumentensouveränität ebenfalls eingeschränkt ist. Der „Preis“, der für die Professionalisierung von OSS zu bezahlen ist, besteht darin, dass die OSS-Entwicklung weniger von enthusiastischen, intrinsisch motivierten Entwicklern, sondern zunehmend auch von kommerziell orientierten Entwicklern (bzw. Firmen, welche für bestimmte Projekte Sponsoring betreiben) getragen wird.

Ein wesentlicher Unterschied zu CSS besteht darin, dass der Wettbewerb nicht direkt bezüglich einzelner Softwareprodukte besteht, sondern bezüglich ganzer Systeme bzw. Komplementärgüter. Zusammen mit der Quelloffenheit und der damit verbundenen offenen Standards führt dies zu einer sehr viel größeren *Vielfalt* an Produkten, Benutzeroberflächen, Konfigurationsmöglichkeiten usw., so dass eine entsprechnde Wahlmöglichkeiten der Kunden besteht. Da wegen der offenen Standards mögliche Wechselkosten zwischen OSS-Produkten gering gehalten werden, sind die tatsächlich ökonomisch nutzbaren Wahlmöglichkeiten – als *ein* Indikator der Konsumentensouveränität – im OSS-Sektor mindestens genauso groß wie im kommerziellen Sektor. Die Vielfalt und der durch offene Standards ermöglichte (sogar erwünschte) Markteintritt neuer Projekte kann sogar dazu führen, dass der Wettbwerb bei der Vermarktung ganzer Systeme oder Komplementärgüter intensiver ist als in monopolisierten Märkten, da keine so ausgeprägte Abhängigkeit des Kunden von einem Hersteller besteht, welcher ein Interesse daran hat, seine Marktmacht nicht nur durch Kundenorientierung, sondern auch durch möglichst hohe Wechselkosten zu sichern. Marktstudien zufolge äußern sich beispielsweise Firmenkunden besorgt über ihre Abhängigkeit vom Software-Marktführer Microsoft (Heise News (2004g)).

Zwar ist es richtig, dass es für spezielle oder extrem anspruchsvolle Kundenwünsche sein kann, dass nicht genügend „freie“ Entwicklerressourcen verfügbar sind, so dass kein entsprechendes OSS-Projekt zustande kommt. Es widerspricht dann aber nicht der Grundidee der Community, dass in solchen Fällen Spezialsoftware oder kundenspezifische Software auch auf kommerziellem Weg erstellt wird. Dabei besteht die Möglichkeit, kommerzielle Software zu entwickeln, die auf OSS-Plattformen ausgeführt werden (z.B. Mathematica oder Maple für Linux).

### *4.5 Wettbewerbswirkungen*

Kommerzielle Unternehmen betrachten inzwischen OSS nicht mehr als Nischenprodukt, sondern als ernst zu nehmende wettbewerbliche Herausforderung (Heise News (2002c) Heise News (2004b) ). Es gibt mehrere direkte und indirekte Kanäle, durch welche die Aktivitäten der Community zu mehr wettbewerblichem Verhalten auf kommerziellen

Softwaremärkten führen und somit zu *deren* Effizienzsteigerung beitragen kann:

- Die OSS-Entwicklung übernimmt Konzepte kommerzieller Software und orientiert sich an deren Eigenschaften, so dass OSS-Produkte auf dem Markt als enges Substitut zu CSS-Produkten gesehen werden und ihnen Marktanteile wegnehmen. Dies führt dazu, dass die starke Abhängigkeit von einem Hersteller verringert wird – schon dies kann von Firmenkunden als strategischer Vorteil gesehen werden, selbst wenn sie dabei Wechselkosten in Kauf nehmen müssen. Manche Kunden, welche ihre Wertschöpfungskette durch den Zukauf proprietärer Software verlängert haben, können dies nun mit OSS-Produkten tun, die möglicherweise besser z.B. auf die angebotene Hardware abstimmbar ist.

- Die OSS-Community entwickelt selbst innovative Konzepte und Standards, die vom kommerziellen Sektor übernommen werden müssen, damit er wettbewerbsfähig bleibt. Wegen restriktiver Open-Source-Lizenzen wie der GPL sind hier dem kommerziellen Sektor Grenzen gesetzt. Dort, wo OSS unter weniger restriktiven Lizenzen verbreitet wird, werden sogar OSS-Lösungen direkt übernommen (z.B. der TCP/IP-Stack in MS-Windows aus der BSD-Linie).

- Das Prinzip der Quelloffenheit führt zu einem transparenten Überprüfen der Software auf Sicherheit und Stabilität durch unzählige Entwickler und Tester weltweit. Mögliche Buffer-Overflows und andere Verwundbarkeiten können zumindest im Prinzip nachvollzogen und entsprechend schnell Maßnahmen getroffen werden. Für eine gefundene Sicherheitslücke kann im Prinzip jeder einen Patch schreiben, dies muss nicht unbedingt der Entwickler des ursprünglichen Codes sein. Dieser sicherheitstechnische Vorzug hat sich in den letzten Jahren als höchst relevant erwiesen und wird auch von kommerziellen Anbietern nachgeahmt, so etwa mit dem erwähnten Shared-Source-Prinzip von Microsoft. Die enormen Ausgaben kommerzieller Anbieter für Softwaresicherheit sind sicherlich auch durch den Wettbewerbsvorsprung von OSS-Produkten bzw. durch den kritischen Vergleich mit OSS zu erklären.

Die Wirkungen der Existenz von OSS auf den Wettbewerb in kommerziellen Softwaremärkten ist quantitativ sehr schwer einschätzbar. Es besteht jedoch die begründete Vermutung, dass der Wettbewerbsdruck im kommerziellen Sektor verstärkt wird bzw. OSS als Substitut für unzureichenden funktionierenden Wettbewerb fungiert.

## 5 Kritische Kommentare zur MICE-Studie

Die Ergebnisse der MICE-Studie werden in sieben Thesen zusammengefasst (vgl. Kalwey u. a. (2003b), S.3 ff.). Vor dem Hintergrund der hier erörterten Zusammenhänge können die Thesen wie folgt kommentiert werden:

- **These 1: Im Kern kein Markt – die Open-Source-Entwicklung**

  Die Studie betont die zentrale Bedeutung preisgesteuerer Märkte für eine effiziente Allokation in einer arbeitsteiligen Ökonomie. Den argumentativen Hintergrund bildet das Modell vollkommener wettbewerblicher Märkte. Dabei wird vernachlässigt, dass es sich bei Software nicht um ein normales Konsumgut handelt: Massive nachfrageseitige Netzwerkexternalitäten, positive Spillovereffekte des in Software steckenden Wissens, sinkende Durchschnittskosten bei hohen first-copy-costs, strategisch beeinflussbare Wechselkosten usw. erfordern weitaus komplexere Marktmodelle. Hier liegen eine Reihe von Ergebnissen in der ökonomischen Literatur vor, dass unter solchen Gegebenheiten nicht unbedingt mit einer effizienten Allokation und maximaler Wohlfahrt zu rechnen ist. Dennoch wird OSS stets dem stilisierten Lehrbuchmodell vollkommenen Wettbewerbs gegenübergestellt, um die angebliche Unterlegenheit gegenüber dem kommerziellen Softwaremarkt zu begründen.

- **These 2: Happy Engineering – Entwicklerorientierung ist nicht Kundenorientierung**

  Dazu wurde in diesem Papier ausführlich argumentiert, weshalb im kommerziellen Softwaremarkt nur in eingeschränktem Umfang der Konsumentensouveränität Rechnung getragen wird, während es im OSS-Bereich Mechanismen der Kundenorientierung (nicht zuletzt durch deren Partizipation am Enstehungsprozess) gibt, welche die Studie nicht ausreichend würdigt. Es wird wieder von einem fiktiven wettbewerblichen Marktmodell ausgehend argumentiert, dass *nur* ein Preissystem in der Lage sei, Konsumentensouveränität durchzusetzen. Es wird dabei vernachlässigt, dass die Preise im kommerziellen Softwaresektor in erheblichem Umfang monopolistische Preissetzungsspielräume aufgrund hoher (und ggf. strategisch erhöhter) Wechselkosten und wegen geringer Markteintrittsmöglichkeiten eine entsprechende Marktmacht widerspiegeln. Da die Grenzkosten einer Softwarelizenz nahezu Null betragen, wird somit in erheblichem Maß Konsumentenrente abgeschöpft. Proprietäre Standards lassen sich als strategisches Instrument verwenden, um die Netzwerkexternalitäten dazu zu nutzen, Konkurrenzprodukten den Marktzutritt zu erschweren, und verringern somit die Wahl- und Wechselmöglichkeiten der Konsumenten. Auf der anderen Seite würdigt die Studie nicht die Mechanismen der Kundenorientierung im OSS-Sektor. Diese beruhen – neben Reputations- und Selbstbindungseffekten – auf dem Interesse an einem etablierten OSS-Umfeld, d.h. an einer Netzwerkgröße, welches die Basis für Komplementärgütervermarktung ist. Dabei wird die Attraktivität des Netzes gerade durch offene Standards und somit dem leichten Marktzutritt unterschiedlicher Softwareprojekte erreicht. Dies erhöht die Vielfalt und Wahlmöglichkeiten der Nutzer bei gleichzeitig verringerten Wechselkosten. Diese Mechanismen schränken die reine Entwicklerorientierung im OSS-Bereich ein.

- **These 3: Nicht kostenlos, aber manchmal umsonst – Die Open-Source-Ressourcenlenkung**

  Auch hier ist das Kernargument der „fehlende Markt", so dass Humankapital mangels Bepreisung nicht effizient auf alternative Verwendungsmöglichkeiten (Softwareprojekte) aufgeteilt werde. Dies gelte insbesondere ex ante bei der Allokation von Entwicklungsressourcen für neue Software. Hier ist entgegenzuhalten, dass in der Community in einem sehr hohen Maß *zusätzliche* Ressourcen für den volkswirtschaftlichen Produktionsprozess akquiriert werden bzw. bestehendes Humankapital *zusätzlich* genutzt, d.h. besser ausgelastet wird. Es wird in der Studie so getan, als bestünde die Opportunität, die Ressourcen auch anderweitig im kommerziellen Sektor zu verwenden. Dies ist nicht der Fall, wenn der Großteil der Entwicklungsarbeit freiwillig im Rahmen der Freizeit erbracht wird. Die Opportunitätsüberlegung der Entwickler ist es dann, freiwillig Entwicklungsbeiträge zu leisten oder Freizeit zu konsumieren. Das Allokationsargument der Studie klammert auch den entscheidenden Aspekt aus, dass es sich bei Software um codiertes Wissen handelt, welches den gesellschaftlichen Wissenskapitaltstock erhöht und positive Spillovers nach sich zieht. Aus der ökonomischen Theorie (z.B. der Neuen Wachstumstheorie) ist jedoch bekannt, dass unter solchen Bedingungen die nur aufgrund von Preissignalen und durch private Anreize vermittelte Ressourcenallokation zu einer Unterversorgung mit solchen Gütern führt. Dies unterstreicht gerade das Steuerungsdefizit des Preissystems. Wenn nun aber Individuen freiwillig Wissensbeiträge leisten und somit eine anreizbedingte Unterversorgung abbauen helfen, ist schwer einzusehen, weshalb dies die Allokationseffizienz verringern sollte. Insgesamt lässt sich festhalten, dass offensichtlich Leistungsanreize, Arbeitsteilung und Koordination nicht allein durch Preise gesteuert werden und Preise weder der einzige Allokationsmechanismus sind, noch Effizienz sicherstellen können.

- **These 4: Second-best Lösung – Kommerzielle Open-Source-Geschäftsmodelle**

  Kommerzielle Komplementärprodukte zu OSS und eine strategische Ausrichtung der Geschäftsmodelle auf OSS werden in der Studie als Verzerrung des Preisgefüges kritisiert: „Kommerzielle Geschäftsmodelle müssen alle Investitionen in Open-Source-Entwicklung indirekt über die Erlöse bei komplementären Produkten finanzieren [...]. Diese Form der Quersubventionierung verfälscht das marktwirtschaftliche Preisgefüge [...]." (Kalwey u. a. (2003b), S.5) Es wird übersehen, dass in kommerziellen Märkten die Lizenzkosten aus Deckungsbeiträgen für (ex post: fixe) Entwicklungskosten und Monopolrenten bestehen und zudem durch monopolistsche Marktstrukturen und strategische Aspekte (z.B. leveraging) geprägt sind. Der Vertrieb ganzer Bündel von Leistungen mit möglichen Quersubventionierungseffekten (insbesondere, wenn es um Fixkostenverteilung geht) ist im kommerziellen Sektor ebenso anzutreffen. Die MICE-Studie geht nicht hinrei-

chend auf den Umstand ein, dass die Grenzkosten eines beliebig replizierbaren Produktes Null sind. Eine Nichtbepreisung von Entwicklerleistung, deren Opportunitätskosten in vielen Fällen in entgangener Freizeit bestehen, aber nicht im Entzug von Ressourcen für alternative Produktionszwecke, ist ebenfalls kein „preisverzerrender“ Tatbestand.

- **These 5: Die Schwächung kommerzieller Software stärkt nicht OSS**

  Die Studie argumentiert, dass der kommerzielle Sektor eine Voraussetzung für das Funktionieren eines OSS-Sektors ist, was jedoch nicht schlüssig belegt wird. Diese These soll hier weder bestritten noch bestätigt werden. Durch die Beitäge zum Wissenskapitalstock und die indirekte Erhöhung des Wettbewerbsdrucks im kommerziellen Sektor wirkt sich aber OSS auch positiv auf diesen aus. Zudem profitiert der kommerzielle Sektor auch von den freiwilligen Akkumulationsleistungen von Humankapital – und sei es nur durch das Aufdecken von Sicherheitlücken in kommerzieller Software durch die Community. Außerdem trägt der OSS-Sektor zu einer Intensivierung des wettbewerblichen Verhaltens im kommerziellen Sektor bei und erhöht so dessen Effizienz. Welcher Bereich vom jeweils anderen stärker profitiert, kann nicht beantwortet werden.

- **These 6: Open-Source-Software fördert nicht den IT-Mittelstand**

  Kaum ein Argument der Studie spiegelt so klar die Geschäftsziele des Auftraggebers der Studie (Microsoft) wider. Unzweifelhaft wirkt sich Microsofts Partnermodell positiv auf den Mittelstand aus und führt dort zu einer hohen Wertschöpfung (vgl. dazu Kalwey u. a. (2003a)). Da der IT-Mittelstand aber lediglich Komplementärprodukte anbietet (darunter auch Software, die auf Microsoft-Produkten aufbaut), könnte er dieses genauso gut für konkurrierende kommerzielle Software oder aber auch für OSS tun. Es gibt eine wachsende Zahl mittelständischer Unternehmen, die mit OSS ihr Geld verdienen. Daher ist diese These schlicht unbegründet. Darüber hinaus ist auch anzumerken, dass jeder Euro, der an Wertschöpfung im IT-Mittelstand erzeugt wird, einen Teil der Total-Cost-of-Ownership (TCO) der IT-Anwender darstellt. Wenn also behauptet wird, dass kommerzielle Software im Mittelstand mehr Wertschöpfung durch Lizenzen und komplementäre Dienstleistungen erzeuge als OSS, so kann das auch so verstanden werden, dass die Software besonders kostenintensiv in puncto Anschaffung, Beratung, Konfiguration, Pflege, Schulung etc. ist.

- **These 7: Die Open-Source-Förderung – Kein Instrument der Wettbewerbspolitik**

  Zunächst ist die Formulierung der These schlicht falsch. OSS-Förderung bzw. OSS-Subventionierung *ist* selbstverständlich ein im Prinzip zur Verfügung stehendes Instrument der Wettbewerbspolitik. Es darf lediglich bezweifelt werden, ob es *sinnvoll* ist, dieses Instrument einzusetzen. Hier ist anzumerken, dass kaum

jemand eine OSS-Subventionierung fordert, insofern zielt diese Warnung völlig ins Leere. Sehr wohl kann man aber das Argument umkehren: Auch die Festlegung der öffentlichen Beschaffungspolitik auf einen (marktführenden) Hersteller wirkt wie eine Subventionierung. Auch die Tatsache, dass die öffentliche Hand die Ausrüstung z.B. von Schulen mit kommerzieller Software eines bestimmten Herstellers zu extrem günstigen Konditionen fördert, wirkt in der Weise, dass spezifisches Humankapital bei den Anwendern aufgebaut wird, so dass sich der Lock-In-Effekt weiter verstärkt und die marktbeherrschende Stellung dieses Anbieters langfristig gesichert wird. Wenn öffentliche Stellen allmählich beginnen, sich in ihrer Beschaffungspolitik von der Festlegung auf den Marktführer zu lösen und OSS als gleichberechtigte Alternative (aus Qualitäts- und Kostengründen) ins Auge zu fassen, wird man wohl kaum von einer Subventionierung und Wettbewerbsverzerrung sprechen können. Als problematisch hingegen ist zu sehen, wenn die Festlegung auf einen bestimmten Hersteller durch die Festlegung auf OSS abgelöst würde, oder feste Quoten vorgeschrieben würden. Der Wettbewerbsdruck, der durch eine stärker differenzierte Beschaffungspolitik erzeugt wird, welche Anschaffungs- und Betriebskosten, Sicherheitsaspekte, Zuverlässigkeit und längerfristige Abhängigkeiten berücksichtigt, kann ordnungspolitisch positiv bewertet werden.

## Literatur


**1 Arrow 1962** Arrow, K. J.: The Economic Implications of Learning By Doing. In: *The Review of Economic Studies* (1962), S. 155–173

**2 Barro und Sala-i Martin 1995** Barro, R. J. ; Martin, X. Sala-i: *Economic Growth.* New York et al. : McGraw-Hill, 1995

**3 Baumol u. a. 1982** Baumol, W. ; Panzar, R. ; Willig, R.: Fixed Cost, Sunk Cost, Entry Barriers and Sustainability of Monopoly. In: *Quarterly Journal of Economics* 95 (1982), S. 405–431

**4 Becker und Michael 1973** Becker, G. S. ; Michael, R. T.: On the New Theory of Consumer Behavior. In: *Swedish Journal of Economics* 75 (1973), S. 378–396

**5 Blankart und Knieps 1992** Blankart, Charles B. ; Knieps, Günter: Netzökonomik. In: Böttcher, Erik (Hrsg.): *Ökonomische Systeme und ihre Dynamik* Tübingen : J.C.B. Mohr (Paul Siebeck), 1992 (Jahrbuch für neue politische Ökonomie)

**6 Choi u. a. 1997** Choi, S. Y. ; Stahl, D. O. ; Whinston, A. B.: Is Microsoft a Monopolist? In: *Brazilian Electronic Journal of Economics* 1 (1997), Nr. 0

**7 Ettrich 2004** Ettrich, M.: Koordination und Kommunikation in Open-Source-Projekten. In: Gehring, Robert A. (Hrsg.) ; Lutterbeck, Bernd (Hrsg.): *Open Source Jahrbuch 2004.* Berlin : Lehmanns Media, 2004, S. 179–192

**8 Ewers u. a. 2003** EWERS, Hans-Jürgen ; FRITSCH, Michael ; WEIN, Thomas: *Marktversagen und Wirtschaftspolitik - mikroökonomische Grundlagen staatlichen Handelns* München : Vahlen, 2003

**9 Fichert 1998** FICHERT, F.: Das Microsoft-Monopol: Herausforderung für die Wettbewerbspolitik. In: *Wirtschaftsdienst* 78 (1998), Nr. 6, S. 343–347

**10 Fichert 2002** FICHERT, Frank: *Wettbewerbspolitik im digitalen Zeitalter - Öffnung vermachteter Märkte virtueller Netzwerkgüter.* Beitrag zum 3. Workshop "Ordnungsökonomik und Recht" des Walter Eucken Instituts. 2002. – www.walter-eucken-institut.de/veranstaltungen/ workshop2002/Fichert-Paper.pdf

**11 Franck und Jungwirth 2002** FRANCK, E. ; JUNGWIRTH, C.: Das Open-Source-Phänomen jenseits des Gift-Society-Mythos. In: *WiSt - Wirtschaftswissenschaftliches Studium* 31 (2002), Nr. 3, S. 124–129

**12 Gandal 1995** GANDAL, Neil: Competing Compatibility Standards and Network Externalities in the PC Software Market. In: *The Review of Economics & Statistics* 77 (1995), Nr. 4, S. 599–608

**13 Gehring 2004** GEHRING, Robert A.: Sicherheit mit Open Source – Die Debatte im Kontext, die Argumente auf dem Prüfstein. In: GEHRING, Robert A. (Hrsg.) ; LUTTERBECK, Bernd (Hrsg.): *Open Source Jahrbuch 2004.* Berlin : Lehmanns Media, 2004, S. 209–236

**14 Gilbert und Katz 2001** GILBERT, Richard J. ; KATZ, Michael L.: An Economist$'$s Guide to U.S. vs. Microsoft. In: *Journal of Economic Perspectives* 15 (2001), Nr. 2, S. 25–44

**15 Grassmuck 2002** GRASSMUCK, Volker: *Freie Software - Zwischen Privat-und Gemeineigentum.* Bonn : Bundeszentrale für politische Bildung ((bpb), 2002. – http://freie-software.bpb.de/Grassmuck.pdf

**16 Gröhn 1999** GRÖHN, Andreas: *Netzwerkeffekte und Wettbewerbspolitik - Eine ökonomische Analyse des Softwaremarktes.* Tübingen : Mohr Siebeck, 1999

**17 Hars und Shaosong 2002** HARS, A. ; SHAOSONG, O.: Working For Free? Motivations For Participating in Open Source Projects. In: *International Journal of Electronic Commerce* 5 (2002), Nr. 3, S. 25–40

**18 Heise News 2002a** HEISE NEWS: *Ältestenrat stimmt für Linux auf Bundestags-Servern* www.heise.de/newsticker/meldung/25701. 14 März 2002

**19 Heise News 2002b** HEISE NEWS: *IBM: Linux rechnet sich.* www.heise.de/newsticker/meldung/24430. 30 Januar 2002

**20 Heise News 2002c** HEISE NEWS: *Microsoft benennt Linux als Konkurrent Nr. 1.* www.heise.de/newsticker/meldung/31661. 19 Oktober 2002

**21 Heise News 2003** Heise News: *HP und IBM scheffeln Geld mit Linux*. www.heise.de/newsticker/meldung/33898. 23 Januar 2003

**22 Heise News 2004a** Heise News: *Linux prescht auf dem Servermarkt vor*. www.heise.de/newsticker/meldung/45061. 27 Februar 2004

**23 Heise News 2004b** Heise News: *Microsoft-Chef sieht Linux als Herausforderung*. www.heise.de/newsticker/meldung/47019. 30 April 2004

**24 Heise News 2004c** Heise News: *Microsoft kann von Linux auch lernen*. www.heise.de/newsticker/meldung/45030. 2004

**25 Heise News 2004d** Heise News: *Open Source: Die kommunale Welt wird bunter*. www.heise.de/newsticker/meldung/46974. 29 April 2004

**26 Heise News 2004e** Heise News: *Open-Source-Projekt erwirkt Verfügung wegen GPL-Verletzung*. www.heise.de/newsticker/meldung/46551. 15 April 2004

**27 Heise News 2004f** Heise News: *Studie: KDE-Entwickler nehmen Benutzerfreundlichkeit ernst*. www.heise.de/newsticker/meldung/46044. 26 März 2004

**28 Heise News 2004g** Heise News: *Studie: Kleinere Unternehmen befürchten Abhängigkeit von Microsoft*. www.heise.de/newsticker/meldung/44157. 29 Januar 2004

**29 Heise News 2004h** Heise News: *Studie übt harte Kritik an Open-Source-Software*. www.heise.de/newsticker/meldung/46544. 15 April 2004

**30 Hertel u. a. 2003** Hertel, G. ; Niedner, S. ; Herrmann, S.: Motivation of Software Developers in Open Source Projects: an Internet-based Survey of Contributors to the Linux Kernel. In: *Research Policy* 32 (2003), Nr. 7, S. 1159–1178

**31 Johnson 2001** Johnson, Justin P.: *Economics of Open Source Software*. Paper. Mai 2001. – http://opensource.mit.edu/papers/johnsonopensource.pdf

**32 Kalwey u. a. 2003a** Kalwey, Nadine ; Kooths, Stefan ; Langenfurth, Markus ; Dieckheuer, Gustav (Hrsg.) ; Kooths, Stefan (Hrsg.): *MICE Economic Research Studies*. Bd. 3: *Die Bedeutung der Microsoft Deutschland GmbH für den deutschen IT-Sektor* Münster : Muenster Institute for Computational Economics, Dez. 2003

**33 Kalwey u. a. 2003b** Kalwey, Nadine ; Kooths, Stefan ; Langenfurth, Markus ; Dieckheuer, Gustav (Hrsg.) ; Kooths, Stefan (Hrsg.): *MICE Economic Research Studies*. Bd. 4: *Open-Source-Software - Eine volkswirtschaftliche Bewertung* Münster : Muenster Institute for Computational Economics, Dez. 2003

**34 Katz und Shapiro 1994** Katz, Michael L. ; Shapiro, Carl: Systems Competition and Network Effects - (Symposia Network Externalities). In: *Journal of Economic Perspectives,* 8 (1994), Nr. 2, S. 93–115

**35 Klemperer 1995** KLEMPERER, P.: Competition When Consumers Have Switching Costs: An Overview With Applications to Industrial Organization, Macroeconomics, and International Trade. In: *Review of Economic Studies* 62 (1995), S. 515–540

**36 Knieps 2001** KNIEPS, Günter: *Wettbewerbsökonomie - Regulierungstheorie, Industrieökonomie, Wettbewerbspolitik.* Berlin : Spinger, 2001

**37 Koglin und Metzger 2004** KOGLIN, O. ; METZGER, A.: Urheber- und Lizenzrecht im Bereich von Open-Source-Software. In: GEHRING, Robert A. (Hrsg.) ; LUTTERBECK, Bernd (Hrsg.): *Open Source Jahrbuch 2004.* Berlin : Lehmanns Media, 2004, S. 293–304

**38 Kogut und Matiu 2001** KOGUT, B. ; MATIU, A.: Open-Source Software Development and Distributed Innovation. In: *Oxford Review of Economic Policy* 17 (2001), Nr. 2, S. 248–264

**39 Kotkamp 2001** KOTKAMP, Stefan: *Electronic Publishing - Ökonomische Grundlagen des Handels mit Informationsprodukten* Karlsruhe, Universität Fridericiana zu Karlsruhe, Dissertation, 2001

**40 Kuhlmann 2004** KUHLMANN, D.: Open Source und offene Standards. In: GEHRING, Robert A. (Hrsg.) ; LUTTERBECK, Bernd (Hrsg.): *Open Source Jahrbuch 2004.* Berlin : Lehmanns Media, 2004, S. 237–248

**41 Leiteritz 2004** LEITERITZ, R.: Open-Source-Geschäftsmodelle. In: GEHRING, Robert A. (Hrsg.) ; LUTTERBECK, Bernd (Hrsg.): *Open Source Jahrbuch 2004.* Berlin : Lehmanns Media, 2004, S. 139–169

**42 Lerner und Tirole 2000** LERNER, Josh ; TIROLE, Jean: *The Simple Economics of Open Source.* März 2000 (NBER working paper series 7600)

**43 Lessig 2002** LESSIG, Lawrence: Open Source Baselines: Compared to What? In: HAHN, Robert W. (Hrsg.): *Government Policy toward Open Source Software* Washington D.C. : AEI-Brookings Joint Center for Regulatory Studies, 2002, S. 50–68

**44 Liebowitz und Margolis 1998** LIEBOWITZ, S. ; MARGOLIS, S.: Network Externality. In: *The New Palgrave Dictionary of Economics and the Law* MacMillan, 1998

**45 Liebowitz und Margolis 1994** LIEBOWITZ, S. J. ; MARGOLIS, S. E.: Network Externality: An Uncommon Tragedy. In: *Journal of Economic Perspectives* 8 (1994), Nr. 2, S. 133–150

**46 Luthiger 2004** LUTHIGER, B.: Alles aus Spaß? Zur Motivation von Open-Source-Entwicklern. In: GEHRING, Robert A. (Hrsg.) ; LUTTERBECK, Bernd (Hrsg.): *Open Source Jahrbuch 2004.* Berlin : Lehmanns Media, 2004, S. 93–106

**47 Mustonen 2003** MUSTONEN, M.: Copyleft – the Economics of Linux and Other Open Source Software. In: *Information Economics and Policy* 15 (2003), Nr. 1, S. 99–122

**48 Nelson 1970** NELSON, Phillip: Information and Consumer Behavior. In: *Journal of Political Economy* 78 (1970), Nr. 2, S. 311–329

**49 Osterloh u. a. 2000** OSTERLOH, M. ; FREY, B.S. ; FROST, J.: Intrinsische Motivation oder residuale Eigentumsrechte? In: *Zeitschrift für Betriebswirtschaft* 70 (2000), Dezember, Nr. 12, S. 1397–1403

**50 Quah 2003** QUAH, Danny: *Digital Goods and the New Economy*. CEPR Discussion Paper No. 3846. März 2003. – http://cep.lse.ac.uk/pubs/download/dp0563.pdf

**51 Shapiro und Varian 2003** SHAPIRO, C. ; VARIAN, H. R.: *Linux Adoption in the Public Sector*. Paper. Dezember 2003

**52 Shapiro und Varian 1999** SHAPIRO, Carl ; VARIAN, Hal R.: *Information Rules - a strategic guide to the network economy*. Boston : Harvard Business School Press, 1999

**53 Stallman 1992** STALLMAN, R.: *Why Software Should Be Free*. www.fsf.org/philosophy/shouldbefree.html. 1992. – Version of April 24, 1992

**54 Stallman 2002** STALLMAN, R. ; GAY, Joshua (Hrsg.): *Free Software, Free Society: The Selected Essays of Richard M. Stallman*. Free Software Foundation, 2002

**55 sueddeutsche.de 2003** SUEDDEUTSCHE.DE, Redaktion: *München setzt auf Linux*. Online Artikel. Mai 2003. – http://www.sueddeutsche.de/muenchen/artikel/128/12116/

**56 Windisch 1987** WINDISCH, Rupert: Privatisierung natürlicher Monopole. In: WINDISCH, Rupert (Hrsg.): *Privatisierung natürlicher Monopole im Bereich von Bahn, Post und Telekommunikation*. Tübingen, 1987

# Jenaer Schriften zur Wirtschaftswissenschaft

# 2004

1/2004 Uwe Cantner, Werner Güth, Andreas Nicklisch, Torsten Weiland: Competition in Innovation and Imitation - A Theoretical and Experimental Study.

2/2004 Uwe Cantner und Andreas Freytag: Eliten, Wettbewerb und langer Atem - Ein praktikabler Vorschlag zur Schaffung von Eliteuniversitäten.

3/2004 Johannes Ruhland und Kathrin Kirchner (Hrsg.): Räumliche Datenbanken - Überblick und praktischer Einsatz in der Betriebswirtschaft.

4/2004 Uwe Cantner und Holger Graf: The Network of Innovators in Jena: An Application of Social Network Analysis.

5/2004 Uwe Cantner and Jens J. Krüger: Empirical Tools for the Analysis of Technological Heterogeneity and Change - Some Basic Building Blocks of "Evolumetrics".

6/2004 Roland Helm: Export Market Entry Strategy and Success: Conceptual Framework and Empirical Examination. *Erscheint als: Market Commitment, Export Market Entry Strategy and Success: Conceptual Framework and Empirical Examination. International Journal of Globalisation and Small Business.*

7/2004 Roland Helm und Michaela Ludl: Kundenkarten als Kundenbindungsinstrument des Handels. *Erscheint als: Kundenbindung im Handel durch Kundenkarten – Determinanten, Wirkungen und Implikationen, Festschrift Prof. Dr. Greipl.*

8/2004 Uwe Cantner, Kristina Dreßler und Jens J. Krüger: Firm Survival in the German Automobile Industry.

9/2004 Marcus Lange und Martin Zimmermann: Patent-Chart - Das Monitoring von Patentportfolios auf der Basis von Zitatbeziehungen

10/2004 Jens J. Krüger: Capacity Utilization and Technology Shocks in the U.S. Manufacturing Sector.

11/2004 Andreas Freytag: EMU Enlargement: Which Concept of Convergence to Apply.

12/2004 Andreas Freytag and Simon Renaud: From Short-Term to Long-Term Orientation – Political Economy of the Policy Reform Process.

13/2004 Martin Kloyer, Roland Helm and Wolfgang Burr: Compensation Preferences of R&D-Suppliers – Some Empirical Results.

14/2004 Roland Helm und Michael Gehrer: Interaktion und Information in der Anbieter-Nachfrager- Beziehung:Voraussetzungen, Konsequenzen und Implikationen der zentralen und peripheren Informationsverarbeitung.

15/2004 Wolfgang Kürsten: Synergies, Shareholder Value and Exchange Ratios in "Value Creating" Mergers - Why Shareholders Should Doubt Management's Pre-Merger Promises.

16/2004 Jens J. Krüger: Using the Manufacturing Productivity Distribution to Evaluate Growth Theories.

17/2004 Andreas Freytag and Klaus Winkler: The Economics of Self-regulation in Telecommunications under Sunset Legislation.

18/2004 Markus Pasche, Sebastian von Engelhardt: Volkswirtschaftliche Aspekte der Open-Source-Softwareentwicklung.

---

Weitere Informationen zu älteren Heften und Bezugsmöglichkeiten: www.wiwi.uni-jena.de Menüpunkt Forschung.