Dieser Artikel ist Teil des

Open Source Jahrbuch 2005

erhältlich unter http://www.opensourcejahrbuch.de.

Das Open Source Jahrbuch 2005 enthält neben vielen weiteren interessanten Artikeln ein Glossar und ein Stichwortverzeichnis.

Open-Source-Software und Standardisierung

Christian Maass und Ewald Scherm

Open-Source-Software steht seit geraumer Zeit im öffentlichen Interesse. Die Auseinandersetzung mit Fragen der Standardisierung erfolgt allerdings nur am Rande. Vielmehr werden damit zusammenhängende Probleme nur oberflächlich dargestellt, oder es kommt zu Fehleinschätzungen. So erachtet man OSS und offene Standards mitunter als Synonyme, obwohl es sich dabei um zwei grundlegend verschiedene Sachverhalte handelt. Vor diesem Hintergrund erfolgt in diesem Beitrag eine Auseinandersetzung mit drei Aspekten, die in Zusammenhang mit Fragen der Standardisierung zwar immer wieder angesprochen, aber nur oberflächlich dargestellt werden: (1) Kompatibilitätsprobleme in Folge des *Forking*, (2) Hersteller(un)abhängigkeit und (3) der Einfluss von Softwarepatenten. Die Auseinandersetzung mit diesen Aspekten soll dazu beitragen, die Diskussion um Open-Source-Software und damit einhergehende Fragen der Standardisierung, differenzierter führen zu können.

1. Ausgangssituation

In den vergangenen Jahren ist Open-Source-Software (OSS) zunehmend in das Interesse der Öffentlichkeit gerückt. Das liegt unter anderem daran, dass dem Anwender von OSS auf Grundlage einer Open-Source-Lizenz sehr weitgehende Nutzungsrechte eingeräumt werden, die im Gegensatz zum „traditionellen" Softwaregeschäft stehen. So kann OSS faktisch kostenlos aus dem Internet bezogen werden, und es ist für den Anwender möglich, den Quellcode – der Aufbau und Arbeitsweise eines Programms offen legt – einzusehen sowie Änderungen daran vorzunehmen. Außerdem ist es gestattet, den Quellcode weiterzugeben, wobei keine Lizenzgebühren erhoben werden dürfen.[1]

Vor diesem Hintergrund betrachtet man OSS als ein öffentliches Gut[2] (Kollock 1999, Bessen 2002, von Krogh 2003), das einen scheinbaren Widerspruch zu dem

1 Vgl. dazu Punkt 1, Free Distribution, der Open Source Definition (Version 1.9), online: http://www.opensource.org/docs/definition.php [25.11.2004].

2 Ein öffentliches Gut ist dadurch gekennzeichnet, dass Personen nicht vom Konsum ausgeschlossen werden können. Gleichzeitig schränkt der Güterkonsum eines Individuums nicht die Konsummöglichkeit anderer Marktteilnehmer ein. Aufgrund dieser Eigenschaften ist es für Unternehmen – die eine Maximierung ihrer Gewinne anstreben – nicht attraktiv, dieses Gut zu produzieren.

in der Ökonomie angenommenen Nutzenmaximierungskalkül darstellt (Franck und Jungwirth 2001). Beiträge, in denen das Phänomen OSS näher betrachtet wird, untersuchen vor allem die Kalküle derer, die OSS entwickeln. Dabei geht es um die Frage, warum Softwareentwickler unentgeltlich ihre Entwicklungsleistung in ein Open–Source-Projekt einbringen (Lerner und Tirole 2000, Hars und Ou 2001, Lakhani und Wolf 2003). Weiterhin betrachtet man den Ablauf der Entwicklung quelloffener Software und damit zusammenhängende organisatorische (Garzarelli 2002, Morner 2002, Demil und Lecocq 2003) und technische Fragen (Koch und Schneider 2002, Scacchi 2002, Shaikh und Cornfold 2003). In diesem Zusammenhang wird in dem Prinzip der quelloffenen Produktentwicklung auch ein neuer Ansatz zur Produktinnovation gesehen (Lerner und Tirole 2001, Chesbrough 2003, von Krogh 2003, Gfaller 2004). Nicht zuletzt ist die Rolle des Staates Objekt einer kritischen Analyse, da sich eine starke Präferenz der öffentlichen Hand für OSS feststellen lässt (Schmidt und Schnitzer 2002, Comino und Manenti 2003).

Die Auseinandersetzung mit Fragen der Standardisierung – worunter man eine Vereinheitlichung nach bestimmten Regeln oder Mustern versteht (Knorr 1993, S. 23) – erfolgt in der wissenschaftlichen Diskussion allerdings nur am Rande. Vielmehr werden damit zusammenhängende Probleme nur oberflächlich dargestellt oder es kommt zu Fehleinschätzungen. So werden OSS und offene Standards im Rahmen einer Informationsbroschüre des Wirtschaftsministeriums des Landes Baden-Württemberg weitestgehend als Synonyme betrachtet (Wirtschaftsministerium 2004, S. 7), obwohl es sich dabei um zwei grundlegend verschiedene Sachverhalte handelt: Während offene Standards darauf abzielen – unabhängig von einem Hersteller – Kompatibilität zwischen den verschiedenen Komponenten eines EDV-Systems herzustellen, geht es bei quelloffener Software um eine spezielle Form der Softwareentwicklung. Open–Source-Lizenzen wie die *GNU General Public Licence* (GPL) sehen dabei vor, dass der Weiterverbreitung des Quellcodes keine Restriktionen in Form von Exklusivitätsklauseln in Lizenzen oder Ansprüchen aus Patenten entgegenstehen, weshalb die Softwareentwicklung konsequenterweise auf offenen Standards beruht. Weiterhin findet man häufig die folgende Auffassung:

> „OSS can help a business or public institution avoid getting locked into a vicious circle of hardware and software upgrades and changes in data formats that require investing in new license fees and significant retraining and can provoke major down time." (United Nations 2003, S. 103)

Auch diese Einschätzung erscheint bei genauerer Betrachtung zu oberflächlich, da Software grundsätzlich den Charakter eines Erfahrungsgutes (vgl. hierzu Abschnitt 3.1.) aufweist, für das Monopolisierungstendenzen nicht ungewöhnlich sind (Nelson 1970). Ferner müssen auch mit der zunehmenden Portierung von unternehmenskritischen Anwendungen auf quelloffene Betriebssysteme, wie z. B. Linux, Investitionen in die Hardware getätigt werden.

Vor diesem Hintergrund sollen in diesem Beitrag drei Aspekte behandelt werden,

die in Zusammenhang mit Fragen der Standardisierung zwar immer wieder angesprochen, aber nur oberflächlich dargestellt werden:

- Kompatibilitätsprobleme in Folge von *Forking*
- Hersteller(un)abhängigkeit und
- Softwarepatente

Die Auseinandersetzung mit diesen Aspekten soll dazu beitragen, die Diskussion um OSS und damit einhergehende Fragen der Standardisierung differenzierter führen zu können.

2. Kompatibilitätsprobleme in Folge von *Forking*

Im so genannten Informationszeitalter kann kaum ein Unternehmen mehr ohne Software im Wettbewerb bestehen. Die Einsatzgebiete reichen von einfachen Schreibhilfen über komplexe Warenwirtschaftssysteme, Server für Datenbanken bis hin zur autarken Steuerung ganzer Produktionsstätten. IT-Abteilungen sehen sich mit der Anforderung konfrontiert, die bestehenden Geschäftsprozesse mit Hilfe der Informationstechnologie abzubilden und damit einen Beitrag zum Unternehmenserfolg zu leisten. Um diesem Ziel gerecht zu werden, ist es erforderlich, die bestehenden Anwendungen an die sich immer schneller ändernden Rahmenbedingungen anzupassen und die Interoperabilität bzw. Kompatibilität der verschiedenen Komponenten der EDV-Systeme zu gewährleisten.

Vor diesem Hintergrund stößt man im Zuge der Auseinandersetzung mit OSS unweigerlich auf das Problem des *Forking*s. Um dieses Problemfeld näher charakterisieren zu können, muss zunächst der Begriff der Kompatibilität präzisiert werden. Ein Großteil der Autoren, die sich mit Fragen der Standardisierung auseinandersetzen, betonen, dass dieser Begriff nur implizit und vage dargestellt wird (David und Bunn 2003, Knorr 1993, Borowicz 2001, Erhardt 2001). Grundsätzlich gelten zwei Güter als kompatibel, wenn sie in bestimmter Weise zusammenarbeiten können (Shy 2001, Martiensen 2004), d. h. Kompatibilität liegt vor, wenn es zwischen den Komponenten eines EDV-Systems möglich ist, über bestimmte Schnittstellen hinweg Informationen auszutauschen (Schmidt und Werle 1994, S. 421). Abhängig von der Beziehung dieser Komponenten untereinander lässt sich zwischen komplementärer und substitutiver Kompatibilität differenzieren; häufig wird auch von direkter und indirekter Kompatibilität gesprochen (Weiber 1992, S. 51, Hecker 1997, S. 38).

Komplementäre Kompatibilität beschreibt die Fähigkeit der Zusammenarbeit ungleichartiger Komponenten. Eine solche Beziehung besteht z. B. zwischen dem Betriebssystem (Z) und darauf aufbauender Anwendungssoftware (X), wenn beide Komponenten die Spezifikation der Schnittstelle (a) einhalten (vgl. Abbildung 1). Erfüllt gleichzeitig die Anwendung (Y) die Spezifikation der Schnittstelle (a), spielt es aus technischer Sicht keine Rolle, ob Software X oder Y verwendet wird, d. h. die Komponenten sind substitutiv kompatibel.

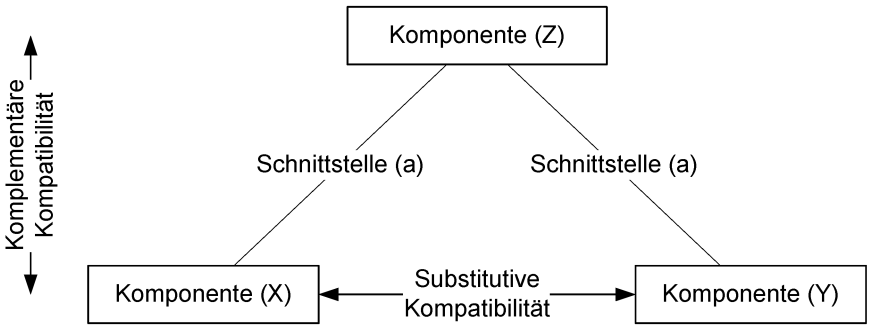

Abbildung 1: Kompatible und substitutive Kompatibilität, in Anlehnung an Pfeiffer (1989, S. 18–25) und Schmidt und Werle (1994, S. 422 f.)

Bezogen auf die Informationstechnik ist der Zweck der Standardisierung darin zu sehen, Kompatibilität zwischen den verschiedenen Komponenten eines Informationsverarbeitungssystems herzustellen (Schmidt und Werle 1994, S. 429)).

Der Begriff *Forking* umschreibt vor diesem Hintergrund die Aufspaltung eines Open-Source-Projektes in verschiedene Programmzweige, die nicht mehr uneingeschränkt kompatibel zum ursprünglichen Standard oder nur auf einen bestimmten Programmzweig abgestimmt sind. Für den Anwender bedeutet dies, dass es zu Störungen der Prozessabläufe kommen kann, mit denen hohe Folgekosten verbunden sind (hierzu auch Ricadela 2001). Zu derartigen Kompatibilitätsproblemen in Folge der Aufteilung eines Open-Source-Projektes kommt es, wenn innerhalb der Entwicklergemeinschaft unterschiedliche Ansichten bezüglich der Projektziele vorliegen (United Nations 2003, S. 102).

Grundsätzlich ist es zwar richtig, dass die Problematik des *Forkings* ein spezifisches Problem der quelloffenen Softwareentwicklung darstellt. Es wird jedoch häufig übersehen, dass sie eng mit der Wahl einer bestimmten Softwarelizenz verbunden ist. Auf diesen Zusammenhang weist beispielsweise (Rosenberg 2000, S. 109–114) hin. Die Gefahr des *Forkings* ist demnach immer dann besonders groß, wenn es aufgrund der Lizenzbestimmungen für Dritte möglich ist, das ursprünglich quelloffene Programm profitorientiert zu vermarkten. Exemplarisch sei an dieser Stelle die *Berkeley Software Distribution* (BSD) genannt, bei dieser Lizenz ist es gestattet, Weiterentwicklungen des Quellcodes als proprietäre Software zu vertreiben. Mit BSD OS, FreeBSD, OpenBSD und NetBSD existiert gleich eine Vielzahl an verschiedenen Programmzweigen des ursprünglich an der Universität Berkeley entwickelten Betriebssystems, wobei der Anreiz zum *Forking* aus kommerziellen Motiven resultiert (Rosenberg 2000, S. 111). Vor diesem Hintergrund befürchtet man eine ähnliche Entwicklung wie im Bereich der Unix-Betriebssysteme in den 80er und 90er Jahren: Hardwarehersteller wie IBM, HP oder DEC boten speziell auf ihre Hardware abgestimmte, proprietäre Unix-Distributionen an, die sich durch spezielle Zusatzfunktionen voneinander unterschieden. Hintergrund dieser Aktivitäten war das Bestreben, durch eine Differenzierungsstra-

tegie Wettbewerbsvorteile aufzubauen. Dies führte letztendlich zu einer Vielzahl von inkompatiblen Unix-Derivaten.

In den so genannten *Copyleft-Lizenzen* sieht man demgegenüber den Vorteil, dass die generelle Verfügbarkeit des Quellcodes die technische Motivation für *Forking* entfernt (Rosenberg 2000, S. 111). Grundsätzlich besteht seitens der Softwarehersteller auch Einigkeit darüber, dass der zukünftige Erfolg quelloffener Betriebssysteme maßgeblich von der Einhaltung eines Referenzstandards abhängt. So erklärte Novells Vizepräsident für das Linux-Geschäft, Jeff Hawkins: „Wir wollen uns durch Support-Angebote und einen höheren Grad an Zuverlässigkeit vom Wettbewerb unterscheiden und nicht durch das Hinzufügen inkompatibler Features" (zitiert nach Cloer und Alexander 2004, S. 4). Unternehmen wie IBM, Hewlett-Packard, Dell, Red Hat, SuSE Linux und Intel haben weiterhin angekündigt, dass sie die im August 2004 veröffentlichte Referenzspezifikationen 2.0 der Linux Standard Base (LSB) unterstützen wollen, um der Gefahr des *Forking* zu begegnen (Cloer 2004).

Die LSB ist eine Initiative der Free Standard Group, die darauf abzielt, die Kompatibilität zwischen den verschiedenen Linux-Distributionen hinsichtlich Dateisystemstruktur und den grundsätzlich notwendigen Programmbibliotheken zu vereinheitlichen, um eine Aufsplitterung in verschiedene Programmzweige zu vermeiden:

> „The LSB Specification is a binary compatibility standard. It specifies, in conjunction with other standards documents which it references, the binary environment in which an LSB compliant application executes. This is different from many computer software standards which define only application programming interfaces (APIs). Such standards apply only to building applications, but do not attempt to provide a cross–platform compatibility where each compliant application's binaries will execute without changes. Specification of the details required to ensure that applications run on each LSB-conformant platform of the same processor type means that interfaces are defined at a binary level." (Base 2004)

Aber nicht nur auf Abnehmerseite führt die Einhaltung dieses Standards zu Vorteilen. Auf Seiten der Softwarehersteller entfallen zeit- und kostenintensive Kompatibilitätstests für die Anwendungssoftware, zumal nicht mehr jede einzelne Linux–Distribution, sondern nur noch ein Standard einem diesbezüglichen Test unterzogen werden muss.[3]

3 Vor dem geschilderten Hintergrund ist auch die seit Monaten andauernde Debatte um die Öffnung des Java-Quellcodes durch Sun zu sehen. Protagonisten der Open-Source-Community, wie Eric Raymond, und Unternehmen, wie IBM, fordern die Quellcode-Freigabe der Programmiersprache, um deren Akzeptanz langfristig zu fördern, zumal es bislang für einige Linux-Distributionen nicht möglich sei, Java als Browser-Plug-In mitzuliefern. Demgegenüber sieht Sun die Gefahr verschiedener und nicht kompatibler Java-Varianten.

3. Hersteller(un)abhängigkeit

3.1. Ursachen der Herstellerabhängigkeit

Im Rahmen der Auseinandersetzung mit Standards gilt es, auf das häufig vorgebrachte Kriterium der Hersteller(un)abhängigkeit näher einzugehen. Um die Ursachen der Hersteller(un)abhängigkeit genauer verstehen zu können, gilt es zunächst,

- die Bedeutung von Netzeffekten zu erläutern und
- Software als Erfahrungsgut zu charakterisieren.

In Zusammenhang mit Netzeffekten wird häufig auch von nachfrageseitigen Skalenerträgen gesprochen, da eine linear steigende Nachfrage zu einer überproportionalen Nutzensteigerung führt (Katz und Shapiro 1986, Wiese 1990). Diese resultiert aus der Zusammenführung kompatibler Komponenten, die bei gemeinsamer Nutzung einen höheren Nutzen stiften als bei isolierter Anwendung. Das lässt sich am „klassischen" Beispiel des Telefonnetzes verdeutlichen (Katz und Shapiro 1985, S. 824). Sofern es noch nicht existiert, wird kein Konsument bereit sein, ein Telefon zu erwerben und Anschlussgebühren zu zahlen. Der Nutzen für ihn wäre gleich Null, da er mit niemandem kommunizieren könnte. Kommt es nun – aus welchen Gründen auch immer – zu einem Anstieg der Teilnehmerzahl, steigt gleichzeitig der Anreiz für potenzielle neue Teilnehmer, sich dem Netzwerk anzuschließen. Besteht das Telefonnetz beispielsweise nur aus den Teilnehmern A und B, existieren nur zwei Kommunikationswege – von A nach B und umgekehrt. Schließt sich ein weiterer Teilnehmer C dem Netzwerk an, erhält jeder der bereits partizipierenden Teilnehmer einen weiteren Kommunikationsweg; insgesamt bestehen dann sechs Verbindungsmöglichkeiten. Ein weiterer Teilnehmer D würde die Anzahl der Kommunikationswege auf zehn erhöhen usw.

Der in diesem Beispiel erläuterte Zusammenhang lässt sich allgemein durch „Metcalfe's Law" beschreiben. Es besagt, dass der Wert eines Netzwerkes – der durch die Anzahl der Kommunikationswege zum Ausdruck kommt – exponentiell mit jedem weiteren Teilnehmer wächst (Shapiro und Varian 1999). Kommt es zu einer derartigen Nutzensteigerung, spricht man von einem *direkten* Netzeffekt.

Zu *indirekten* Netzeffekten kommt es dagegen bei komplementären Komponenten. Eine solche Verbindung besteht z. B. zwischen Betriebssystem und Anwendungssoftware (Achi et al. 1995, S. 8). So stellen Softwareunternehmen aufgrund des hohen Verbreitungsgrades des Microsoft-Betriebssystems bevorzugt darauf basierende Applikationen her, um ein möglichst großes Kundensegment ansprechen zu können. Für Käufer wiederum ist die Zahl der verfügbaren Komplementärprodukte ein ausschlaggebendes Kaufkriterium für Windows, weshalb sich die Anwenderzahl wiederum erhöht und weitere Softwareunternehmen Windows-basierte Anwendungen herstellen. Erwirbt ein Konsument eine zusätzliche Einheit des Basisproduktes, steigt gleichzeitig der Anreiz für einen Komplementärguthersteller, hierauf aufbauende Anwendungen herzustellen. Durch das größere Sortiment an komplementären Produkten wird indi-

rekt der Nutzen der anderen Nutzer gesteigert. Insofern führt die starke Verbreitung eines Basisproduktes zu einem reichhaltigen Angebot an Komplementärprodukten.

Der Einfluss von Netzeffekten auf den Wettbewerb führt schließlich dazu, dass das Produkt mit dem höchsten Verbreitungsgrad überproportional bevorzugt wird. Ein Wettbewerb zwischen zwei Technologien findet deshalb nur so lange statt, bis sich der Markt für eine Technologie entschieden hat (Arthur 1989). Dabei kommt es häufig – wie im Fall des Microsoft Betriebssystems – zur Herausbildung eines (defacto-)Standards. Hierunter versteht man eine Technologie, die sich in Folge einer breiten Akzeptanz auf der Nachfrageseite im Wettbewerb durchsetzen konnte (Katz und Shapiro 1994, S. 105). In dieser Situation verfügt das Unternehmen über einen monopolistischen Preissetzungsspielraum (Hess 2000, S. 97), der zumindest vorübergehend eine überdurchschnittliche Rendite erwarten lässt. Gleichzeitig kommt es aber auch zu einem Innovationsproblem (Farrell und Saloner 1985), da neue Produkte im Vergleich zu den etablierten nur einen geringen Netznutzen aufweisen und die Abnehmer in Folge der Netzeffekte bei dem etablierten Standard verharren. Aus diesem Grund fällt es schwer, eine neue Technologie – selbst wenn diese qualitativ, z. B. hinsichtlich der Funktionalität, überlegen ist – in den Markt einzuführen.

Weiterhin gilt es, in der Auseinandersetzung mit dem Kriterium der Hersteller(un)abhängigkeit zu beachten, dass Software grundsätzlich ein Wirtschaftsgut darstellt, das durch einen hohen Komplexitätsgrad gekennzeichnet ist (Balzert 2000, S. 29 f.). In der Regel ist es deshalb für den Anwender erst nach einer gewissen Einarbeitungszeit möglich, die Software operativ zu nutzen. Insofern stellt Software ein Erfahrungsgut dar, d. h. erst nach einer gewissen Zeit ist der Anwender in der Lage, sich ein Bild über die Eignung des Produktes zu machen (Nelson 1970, S. 327). Bis dahin fallen neben den Anschaffungsausgaben Einrichtungs- und Lernkosten an, die erstere deutlich übersteigen können. Aus diesem Grund gestaltet sich der Wechsel zu einer anderen Softwarelösung häufig als problematisch. Hinzu kommt, dass die bis dahin getätigten Ausgaben irreversible Investitionen darstellen, die bei einem Produktwechsel „verloren" gehen. Nicht zuletzt aus diesen Gründen zeigen Erfahrungsgüter eine Tendenz zur Monopolbildung auf (Nelson 1970, S. 327).

Mit der Kenntnis von Netzeffekten und der Kennzeichnung von Software als ein Erfahrungsgut ist es möglich, das Kriterium der Hersteller(un)abhängigkeit genauer zu analysieren.

3.2. Open Source und Hersteller(un)abhängigkeit

Ein aktuelles Beispiel für die Auswirkungen einer Abhängigkeit im Bereich der proprietären Software ist die Beendigung der Herstellerunterstützung für Windows NT durch den Hersteller Microsoft. Die Anwender stehen vor der Wahl, auf weitere Fehlerbehebung und Support zu verzichten oder ein Update auf Windows 2000 vorzunehmen. Das Update ist nicht nur mit direkten Kosten in Form des Erwerbs neuer Lizenzen verbunden, es fallen gegebenenfalls auch noch Folgekosten an. Zum Beispiel stellt der Betrieb von Windows 2000 höhere Anforderungen an die Hardware und aufgrund der Umstellung von dem NT-Domänenkonzept auf *Windows 2000/Active Directory*

sind strukturelle Anpassungen notwendig. In Folge des hohen Verbreitungsgrades des Microsoft-Betriebssystems und der daraus resultierenden Netzeffekte kann das Unternehmen hohe Preise durchsetzen, zumal die Anwender nicht ohne Weiteres den Standard wechseln können, wenn sie auf den problemlosen Datenaustausch mit ihren Geschäftspartnern angewiesen sind. Es verwundert daher nicht, dass der Vorstandsvorsitzende von Microsoft, Steve Ballmer, den Erhalt der installierten Basis als „größte Herausforderung" ansieht (Holzwart 2004).

Der Vorteil einer quelloffenen Implementierung wird demgegenüber in der Herstellerunabhängigkeit gesehen (United Nations 2003, S. 103). Da hier grundsätzlich keine Lizenzkosten anfallen, besteht ein Kostenvorteil auf Seiten der quelloffenen Software. Allerdings stellen diese Lizenzkosten nur einen Teil der Gesamtkosten des Softwareeinsatzes dar. So fallen auch bei OSS durch kontinuierliche Fehlerbehebung, Implementierung neuer Funktionen bis hin zu neuen Designs Veränderungen an, die – wie für ein Erfahrungsgut typisch – einen entsprechenden Schulungs- und Installationsaufwand nach sich ziehen, woraus eine gewisse Produktbindung entsteht. Dabei spielt es keine Rolle, ob ein proprietärer oder offener Standard unterstützt wird. Mit der zunehmenden Portierung unternehmenskritischer Anwendungen auf Linux gilt es zudem, Kosten für leistungsfähigere Hardware zu berücksichtigen.

Der „Unterschied" zwischen den beiden Softwarekonzeptionen – bezogen auf das Kriterium der Hersteller(un)abhängigkeit – lässt sich vor dem geschilderten Hintergrund im Wesentlichen anhand von zwei Punkten verdeutlichen:

- Zeitpunkt des Updates
- Wahl eines Herstellers

Ein wichtiger Vorteil der quelloffenen Software wird darin gesehen, dass der Anwender selbst über den Zeitpunkt bestimmen kann, wann ein Update installiert oder eine Hardware-Investition getätigt werden soll (Herrmann 2004). Diese Freiheit besteht im Bereich der proprietären Software nur bedingt. Exemplarisch seien hier die restriktiv gestalteten Lizenzverträge von Microsoft genannt. Im Rahmen des so genannten „Software Assurance" Programms können Anwender zwar Updates zu vergünstigten Konditionen erwerben, allerdings sind sie mehr oder weniger dazu gezwungen, zumal Microsoft keine Updates für ältere Programmversionen zur Verfügung stellt. Mit dieser Lizenzpolitik – die auf Abnehmerseite auch auf massiven Protest gestoßen ist (Krempl 2002) – will Microsoft erreichen, dass die Anwender regelmäßig neue Softwarelizenzen erwerben und ihre Software auf dem aktuellen Stand halten.

Allerdings stellt sich in dieser Hinsicht auch die Lizenzpolitik der etablierten Linux–Distributoren, wie Red Hat und Novell/SuSE, als nicht unproblematisch dar. Um beispielsweise neue Updates der SuSE-Distribution installieren zu können, müssen die Anwender eine so genannte „Upgrade Protection" erwerben, die jährlich erneuert werden muss (Novell 2002, S. 6):

> „Customers can deploy Novell open source products by adding Upgrade Protection or Maintenance to cover their total usage. Upgrade

> Protection is a component of the Volume Licence Agreement (VLA) and Corporate License Agreement (CLA), which ensures customers always have the latest updates to their products. [. . .] Novell open source technologies require the purchase of upgrade protection for each server-/workstation where the product is installed." (Novell 2004, S. 23).

Der Erwerb einer solchen Upgrade Protection ist ferner auch Voraussetzung, um sich für das so genannte „Indemnification Program" zu qualifizieren, bei dem Anwender vor potenziellen Rechtsansprüchen Dritter – gesetzt den Fall, die Open-Source–Lösung verstößt gegen urheberrechtlich geschützten Quellcode – abgesichert werden sollen. Insofern besteht auch bei den etablierten Anbietern von Linux-Distributionen ein Zwang, kontinuierlich neue „Softwarepflegeverträge" abzuschließen. Die Freiheit, selbst über den Zeitpunkt des Softwareupdates zu entscheiden, ist deshalb lediglich im Bereich „alternativer" Linux-Distributionen, wie z. B. Debian, gegeben, die durch die Open-Source-Community entwickelt werden und bei denen derartige Pflegeverträge nicht erforderlich sind (Herrmann 2004).

Die Implementierung offener Standards führt weiterhin dazu, dass mitunter eine Vielzahl von Unternehmen in den Wettbewerb tritt. Dies begünstigt die Verhandlungsstärke der Anwender, die aus einem größeren Produktangebot wählen können. Dabei gilt es jedoch zu beachten, dass die beiden Konzepte „offener Standard" und „Open Source" nicht synonym verwendet werden dürfen, zumal auf Grundlage eines offenen Standards auch proprietäre Softwarelösungen auf dem Markt eingeführt werden, die mitunter sogar einen weitaus höheren Marktanteil als die quelloffene Software aufzeigen. Exemplarisch sei an dieser Stelle der Bereich der Applikationsserver genannt, bei dem die etablierten proprietären Produkte von IBM, BEA und Novell mit quelloffenen Softwarelösungen, wie z. B. Zope oder JBoss, konkurrieren. Bezogen auf die Hersteller(un)abhängigkeit ist es somit zu pauschal, offene Standards und Open Source als Synonyme zu betrachten.

4. Softwarepatente und offene Standards

4.1. Rechtliche Rahmenbedingungen

Die Auseinandersetzung mit offenen Standards und OSS zeigt nicht zuletzt ein Problemfeld, das sich aus Softwarepatenten ergibt, die der EU-Wettbewerbsrat gegenwärtig auf die Softwareentwicklung übertragen will (Europäische Kommission 2004). Dadurch sollen die Schutzmöglichkeiten von Software erweitert werden. So ist es beispielsweise nach dem Urheberrecht lediglich möglich, nur eine bestimmte Programmimplementierung – nicht aber das Verfahren bzw. die Idee als solche – zu schützen. Bei einem Patent lassen sich demgegenüber auch die hinter der Software stehenden Ideen schützen, wodurch der Patentinhaber ein gegenüber Dritten wirksames Ausschließlichkeitsrecht erhält. Gesetzlich wird die Patentierung von Software durch die Europäische Patentübereinkunft (EPÜ) geregelt, die bislang ein grundsätzliches Patentierungsverbot so genannter „Programme für Datenverarbeitungsanlagen" vorsieht (EPÜ Artikel 52, Absatz 2). Allerdings wurden seit Anfang der 1980er Jahre

bereits mehr als 30 000 Softwarepatente angemeldet, indem Software als „wesensbestimmender Bestandteil" eines technischen Geräts interpretiert wurde (vgl. Maaß und Scherm 2004, S. 1192). Der Zusammenhang zwischen Softwarepatenten, offenen Standards und OSS soll im Folgenden anhand des Mono-Projektes verdeutlicht werden.

4.2. Fallbeispiel: Mono

Wie eingangs bereits erläutert, spielt die Interoperabilität zwischen den verschiedenen Komponenten eines EDV-Systems eine immer wichtigere Rolle. In diesem Zusammenhang gewinnen so genannte Web Services zunehmend an Bedeutung. Darunter versteht man Programmkomponenten, die von anderen Programmen über das Internet in Anspruch genommen werden. Die einzelnen Programmkomponenten lassen sich auf der Basis standardisierter Datenformate und Internetprotokolle – wie z. B. XML oder TCP/IP – auf unterschiedliche Art und Weise miteinander kombinieren. Unter Verwendung dieser Schnittstellen können auch heterogene Systeme miteinander kommunizieren. Insofern erlauben Web Services eine flexible Modellierung und Anpassung der Geschäftsprozesse an veränderte Rahmenbedingungen (Beimborn und Weitzel 2003, S. 1360).

Gegenwärtig existieren zwei „große" Plattformen, um derartige Anwendungen zu entwickeln. Auf der einen Seite handelt es sich dabei um die *Java 2 Enterprise Edition* (J2EE). J2EE ist ein offener Standard, der auf der proprietären Programmiersprache Java von Sun beruht, deren Vorteil insbesondere in der Plattformneutralität gesehen wird, d. h. J2EE ist nicht an ein bestimmtes Betriebssystem gebunden. Auf der anderen Seite gewinnt auch das .NET-Framework von Microsoft zunehmend an Bedeutung, für die Entwicklung von webbasierten Anwendungen. Im Gegensatz zu J2EE konnte dieses Framework in der Vergangenheit allerdings ausschließlich auf Rechnern genutzt werden, die mit dem Betriebssystem Windows ausgestattet waren. Erschwerend kam hinzu, dass Microsoft Funktionen des Frameworks mit dem Betriebssystem verknüpft hat, die „traditionell" von Applikationsservern übernommen werden.

Vor diesem Hintergrund entstand das Mono-Projekt, um eine quelloffene Implementierung des .NET-Frameworks unter Linux zu realisieren. Das Interesse der Open-Source-Community begründet vor allem in der Tatsache, dass sich – im Gegensatz zu J2EE, das auf der Sprache Java basiert – unter .NET eine Vielzahl an Programmiersprachen bei der Softwareentwicklung kombinieren lässt. Die unterstützten Sprachen greifen zu diesem Zweck auf einheitliche Klassenbibliotheken und Schnittstellen zurück. Da die verschiedenen Programmiersprachen für bestimmte Probleme mehr oder weniger geeignet sind, lassen sich so die Stärken der jeweiligen Sprache im Rahmen der Softwareentwicklung kombinieren.

Die Voraussetzung für das Mono-Projekt wurde faktisch durch Microsoft selbst geschaffen, indem es die betreffenden Technologien bei der *European Computer Manufacturers Association* (ECMA) und der *International Organization for Standardization* (ISO) zur Standardisierung einreichte. Damit wurden die Funktionalitäten des .NET-Frameworks spezifiziert und die Standards sind frei zugänglich und nicht

diskriminierend. Das Problem bei der Open-Source-Implementierung ist allerdings darin zu sehen, dass die durch ECMA und ISO standardisierten Technologien nur einen Teil der .NET-Technologie abdecken. Dabei handelt es sich um die Sprache C# und die *Common Language Infrastructure* (CLI), welche die Laufzeitumgebung des .NET-Frameworks spezifizieren. Im Rahmen des Mono-Projekts werden aber auch darauf aufbauende Technologien berührt, wie z. B. ADO.NET[4] und ASP.NET[5]. Diese Technologien sind immer dann von hoher Bedeutung, wenn Anwender vollständige Kompatibilität zu Windows-Plattformen benötigen. Im Rahmen des Mono-Projekts wurden diese und andere Technologien faktisch „nachimplementiert" (Mono 2004). Verstöße gegen damit einhergehende Softwarepatente stellen allerdings ein nicht abschätzbares Risiko dar, weshalb die Erfolgsaussichten des Projektes nach wie vor umstritten sind. Mitunter wird befürchtet, dass das Entwicklungsprinzip quelloffener Software durch Patente auf Algorithmen unterlaufen und im Fall der Verletzung proprietärer Softwarepatente lizenzpflichtig werden könnte (Mueller 2001).

Im Rahmen des Mono-Projektes wird auf die patentrechtlichen Probleme auf dreierlei Weise reagiert:

> „(1) work around the patent by using a different implementation technique that retains the API, but changes the mechanism; if that is not possible, we would (2) remove the pieces of code that were covered by those patents, and also (3) find prior art that would render the patent useless." (Mono 2004)

Der Implementierung eines offenen Standards können somit patentrechtliche Probleme entgegenstehen, zumal Open-Source-Lizenzen wie die GPL die Einbindung patentrechtlich geschützter Technologien explizit untersagen. Neben den Auswirkungen auf die quelloffene Softwareentwicklung zeigen sich ähnlich gelagerte Probleme für mittelständische Software-unternehmen, die nicht über die finanziellen Mittel verfügen, um im Vorfeld umfangreiche Patentrecherchen anzustellen oder ein langwieriges Patentierungsverfahren zu bestreiten (Jakob 2003, S. 6–7). Patentrechtliche Probleme stellen insofern zwar kein spezifisches Problem quelloffener Software dar, allerdings hat diese den Nachteil, dass eine Überprüfung auf Patentverletzungen durch die Zugriffsmöglichkeit auf den Quellcode sehr viel einfacher als im Fall proprietärer Software ist.

Bei dem anhand des Mono-Projektes aufgezeigten Konflikt zwischen offenen Standards und Open-Source-Software handelt es sich nicht um einen Einzelfall. Aufsehen erregte beispielsweise der Patentstreit zwischen Kodak und Sun. Ein New Yorker Gericht bestätigte, dass Suns Programmiersprache Java Patente von Kodak verletzte (Vaske 2004). Die Unternehmen einigten sich auf einen Vergleich in Höhe von 92 Millionen US-Dollar. Gleichzeitig erklärte sich Sun dazu bereit, Lizenzen für die betreffenden Technologien zu erwerben. Es ist wichtig anzumerken, dass durch diese

4 Durch ADO.NET werden Dienste für den Datenbankzugriff bereitgestellt.

5 ASP.NET stellt die .NET-Komponente für die Entwicklung von Web Services dar und ist der Nachfolger von *Active Server Pages* (ASP).

Übereinkunft auch Lizenznehmer von Sun „geschützt" sind. Neben IBM und Bea zählen hierzu auch die Open-Source-Projekte JBoss, Geronimo und Objectweb.

5. Fazit

Fragen der Standardisierung spielen im Rahmen der quelloffenen Softwareentwicklung eine bedeutende Rolle. Bislang wurden damit einhergehende Fragen nur am Rande behandelt. Angesichts dessen ging dieser Beitrag auf drei Punkte ein, die häufig zu Missverständnissen führen bzw. nur oberflächlich dargestellt werden.

Erstens handelt es sich um das so genannte *Forking* und damit einhergehende Kompatibilitätsprobleme. Im Rahmen etablierter Open-Source-Projekte, wie z. B. im Fall von Linux, wird dieser Problematik dadurch begegnet, dass sich die führenden Softwareunternehmen auf einen Referenzstandard geeinigt haben, um die Kompatibilität der verschiedenen Programmzweige zu gewährleisten. Es bleibt abzuwarten, inwieweit diese Spezifikationen in Zukunft eingehalten und umgesetzt werden.

Zweitens wurde auf den Aspekt der Hersteller(un)abhängigkeit eingegangen. Es stellte sich heraus, dass Software – verstanden als Netzeffekt- und Erfahrungsgut – grundsätzlich eine Tendenz zur Monopolisierung aufweist. Sowohl im Bereich der quelloffenen als auch der proprietären Software besteht durch die Lizenzpolitik vieler Hersteller ein gewisser Zwang, regelmäßig neue Updates zu beziehen. Lediglich bei „alternativen" Linux-Distributionen besteht nach wie vor die Freiheit, selbst über den Zeitpunkt eines Updates zu entscheiden. Der Vorteil eines offenen Standards wurde vor diesem Hintergrund darin gesehen, dass der Anwender aus einer Vielzahl von Anbietern wählen kann, die sich selbst in einem intensiven Wettbewerb untereinander befinden. Dabei kann es sich sowohl um proprietäre als auch quelloffene Software handeln.

Drittens wurden Softwarepatente angesprochen, die die Implementierung eines offenen Standards behindern, zumal Open-Source-Lizenzen die Einbindung proprietärer bzw. patentrechtlich geschützter Technologien explizit untersagen. Allerdings stellen Softwarepatente kein spezifisches Problem quelloffener Software dar. So stellt es sich beispielsweise für mittelständische Softwareunternehmen als problematisch dar, wenn diese unwissentlich gegen ein Patent verstoßen bzw. nur über begrenzte finanzielle Mittel verfügen, um im Vorfeld umfangreiche Patentrecherchen anzustellen oder ein langwieriges Patentierungsverfahren zu bestreiten.

Wie sich die Standardisierungsbemühungen im Open-Source-Bereich in Zukunft entwickeln werden, lässt sich nicht eindeutig prognostizieren. Dies liegt nicht zuletzt daran, dass gerade in diesem Zusammenhang im politischen Umfeld widersprüchliche Entscheidungen getroffen werden. Beispielsweise positioniert sich die öffentliche Hand auf der einen Seite als Förderer der quelloffenen Softwareentwicklung. Auf der anderen Seite wird wiederum die Einführung von Softwarepatenten diskutiert, die – wie bereits dargestellt – im Gegensatz zum Entwicklungsprinzip der quelloffenen Software stehen.

Literaturverzeichnis

Achi, Z., Doman, A., Sibony, O., Sinha, J. und Witt, S. (1995), 'The paradox of fast growth tigers', *The McKinsey Quarterly* **3**, S. 4–17.

Arthur, W. B. (1989), 'Competing Technologies, increasing returns, and lock-in by historical events', *Economic Journal* **97**, S. 642–665.

Balzert, H. (2000), *Lehrbuch der Software-Technik*, 2. Aufl., Spektrum, Heidelberg, Berlin.

Base, L. S. (2004), 'Project FAQ'. http://www.linuxbase.org/modules.php?name=FAQ [19. Okt 2004].

Beimborn, D. und Weitzel, T. (2003), 'Web-Services und Service-orientierte Architekturen', *Das Wirtschaftsstudium* **32**, S. 1360–1364.

Bessen, J. (2002), 'Open Source Software: Free Provision of Complex Public Goods', http://opensource.mit.edu/ [20. Jul 2004].

Borowicz, F. (2001), *Strategien im Wettbewerb um Kompatibilitätsstandards*, Europäische Hochschulschriften, Lang, Frankfurt am Main. Reihe 5, Volks- und Betriebswirtschaft. Bd. 2802.

Chesbrough, H. (2003), 'The Era of Open Innovation', *Sloan Management Review* **44**(3), S. 35–41.

Cloer, T. (2004), 'LSB 2.0 soll Linux gegen Microsoft stärken', *Computerwoche* . http://www.computerwoche.de/index.cfm?pageid=254&artid=65075&main_id=6507 [19. Okt 2004].

Cloer, T. und Alexander, S. (2004), 'Unix-Schicksal soll Linux erspart bleiben', *Computerwoche* **31**.

Comino, S. und Manenti, F. (2003), 'Open Source vs Closed Source Software: Public Policies in the Software Market', http://opensource.mit.edu/ [20. Jul 2004].

David, P. und Bunn, J. (2003), 'The economics of Gateway Technologies and Network Evolution: Lessons from electricity supply history', *Information Economics and Policy* **3**, S. 165–202.

Demil, B. und Lecocq, X. (2003), 'Neither Market nor Hierarchy or Network: The Emerging Bazaar Governance', http://opensource.mit.edu/ [20. Jul 2004].

Erhardt, M. (2001), *Netzwerkeffekte, Standardisierung und Wettbewerbsstrategie*, Gabler / DUV, Wiesbaden.

Europäische Kommission (2004), 'Patentierbarkeit computerimplementierter Erfindungen'. http://europa.eu.int/comm/internal_market/de/indprop/comp/index.htm [10. Aug 2004].

Farrell, J. und Saloner, G. (1985), 'Standardization, Compatibility, and Innovation', *Rand Journal of Economics* **16**, S. 70–83.

Franck, E. und Jungwirth, C. (2001), 'Open versus Closed Source – Eine organisationsökonomische Betrachtung zum Wettbewerb der Betriebssysteme Windows und Linux'. http://www.isu.unizh.ch/fuehrung/Dokumente/WorkingPaper/4full.pdf.

Garzarelli, G. (2002), 'Open Source Software and the Economics of Organization'. http://opensource.mit.edu/ [20. Jul 2004].

Gfaller, H. (2004), 'Open Source und Linux – Von wegen innovativ'. ZDNet Deutschland http://www.zdnet.de/itmanager/kommentare/0,39023450,39122696,00.htm [20. Jul 2004].

Hars, A. und Ou, S. (2001), Working for Free? Motivations of Participating in Open Source Projects, *in* 'Proceedings of the 34th Hawaii International Conference on System Sciences'.

Hecker, F. (1997), Die Akzeptanz und Durchsetzung von Systemtechnologien – : Marktbearbeitung und Diffusion am Beispiel der Verkehrstelematik, Master's thesis, Universität des Saarlandes, Saarbrücken. XVII, 257 S., Diss. 1998, – Prof. Dr. Joachim Zentes.

Herrmann, W. (2004), 'Wir brauchen keinen Servicevertrag', Computerwoche, http://www.computerwoche.de/index.cfm?pageid=255&artid=65730&type=detail&kw=Wir%20brauchen%20keinen%20Servicevertrag [19. Okt 2004].

Hess, T. (2000), 'Netzeffekte: Verändern neue Informations- und Kommunikationstechnologien das klassische Marktmodell?', *Wirtschaftswissenschaftliches Studium* **29**.

Holzwart, G. (2004), 'Microsoft: Viel Cash und wenig Innovationen', Computerwoche, http://www.computerwoche.de/index.cfm?pageid=258&artid=60571&main_id=60571&category=8&currpage=1&type=detail&kw=steve%20ballmer%20installierte%20basis [22. Sep 2004].

Jakob, G. (2003), 'Positionspapier des Vereins zur Förderung Freier Software zum gegenwärtigen Stand der Diskussion um Logik- und Ideenpatente', http://ffs.or.at/Mitglieder/jack/position_swp.pdf.

Katz, M. L. und Shapiro, C. (1985), 'Network externalities, competition, and compatibility', *American Economic Review* **75**, S. 424–440.

Katz, M. L. und Shapiro, C. (1986), 'Technology Adoption in the Presence of Network Externalties', *Journal of Political Economy* **94**, S. 822–841.

Katz, M. L. und Shapiro, C. (1994), 'Systems competition and network effects', *Journal of Economic Perspectives* **8**, S. 93–115.

Knorr, H. (1993), *Ökonomische Probleme von Kompatibilitätsstandards. Eine Effizienzanalyse unter besonderer Berücksichtigung des Telekommunikationsbereichs*, Nomos, Baden-Baden.

Koch, S. und Schneider, G. (2002), 'Effort, co-operation and co-ordination in an open source software project: GNOME', *Information System Journal* **12**, S. 27–42.

Kollock, P. (1999), The economies of online cooperation: Gifts and public goods in cyberspace, *in* M. A. Smith und P. Kollock (Hrsg.), 'Communities in Cyberspace', Routledge, London, Kapitel 9, S. 220–39.

Krempl, S. (2002), 'Wachsende Empörung über Microsofts neue Lizenzpolitik', http://www.heise.de/newsticker/meldung/28860 [22. Sep 2004].

Lakhani, K. und Wolf, R. (2003), 'Why Hackers do what they do: Understanding Motivation Effort in Free/Open Source Software Projects', http://ssrn.com/abstract=443040 [20. Jul 2004]. MIT Sloan School of Management, Working Paper 4425-03.

Lerner, J. und Tirole, J. (2000), 'The Simple Economics of Open Source', http://www.nber.org/papers/w7600 [19. Okt 2004]. National Bureau of Economic Research, Working Paper 7600.

Lerner, J. und Tirole, J. (2001), 'The Open Source Movement: Key Research Questions', *European Economic Review* **45**, S. 819–826.

Maaß, C. und Scherm, E. (2004), 'Softwarepatente', *Das Wirtschaftsstudium* **33**, S. 1192 ff.

Martiensen, J. (2004), 'Netzwerkökonomie'. Diskussionsbeitrag des Fachbereichs Wirtschaftswissenschaften der Fernuniversität Hagen; Nr. 358.

Mono (2004), *Frequently Asked Questions*, Mono-Projekt. http://www.mono-project.com/about/licensing.html [19. Okt 2004].

Morner, M. (2002), 'Das Open-Source-Software-Phänomen – organisatorisch betrachtet', *Zeitschrift Führung und Organisation* **71**, S. 219–225.

Mueller, D. (2001), 'Fängt Microsoft Open Source in seinem .NET?', ZDNet, http://www.zdnet.de/news/specials/webservice/0,39023676,2092702,00.htm [19. Okt 2004].

Nelson, P. (1970), 'Information and Consumer Behavior', *Journal of Political Economy* **78**, S. 311–329.

Novell (2002), 'Novell Lizenzvereinbarungen'.

Novell (2004), 'Novell's One Net Strategy'.

Pfeiffer, G. (1989), *Kompatibilität und Markt: Ansätze zu einer ökonomischen Theorie der Standardisierung*, Nomos, Baden-Baden.

Ricadela, A. (2001), 'The State of Software Quality', http://www.information-week.com/838/quality_poll.htm [10. Feb 2004].

Rosenberg, D. K. (2000), *Open Source – The Unauthorized White Papers*, M&T Books, Foster City.

Scacchi, W. (2002), Is Open Source Software Development Faster, Better, and Cheaper than Software Engineering?, *in* 'ICSE Workshop on Open Source Software Engineering', Orlando.

Schmidt, K. und Schnitzer, M. (2002), 'Public Subsidies for Open Source? Some Economic Policy Issues of the Software Market', http://opensource.mit.edu/ [20. Jul 2004].

Schmidt, S. und Werle, R. (1994), Die Entwicklung von Kompatibilitätsstandards in der Telekommunikation, *in* M. Tietzel (Hrsg.), 'Ökonomik der Standardisierung', Accedo, München, S. 419–448.

Shaikh, M. und Cornfold, T. (2003), 'Version Management Tools: CVS to BK in the Linux Kernel', http://opensource.mit.edu/ [20. Jul 2004].

Shapiro, C. und Varian, H. R. (1999), *Information Rules: A Strategic Guide to the Network Economy*, Harvard Business School Press, Boston.

Shy, O. (2001), *The Economics of Network Industries*, Cambridge University Press, Cambridge, UK.

United Nations (2003), *E-Commerce and Development Report 2003*, United Nations, New York, Genf. http://www.unctad.org/en/docs/ecdr2003ch4_en.pdf [4. Sep 2004].

Vaske, H. (2004), 'Kodak und Sun einigen sich im Patentstreit', Computerwoche, http://www.computerwoche.de/index.cfm?pageid=254&artid=65964 [19. Okt 2004].

Weiber, R. (1992), *Diffusion von Telekommunikation*, Dr. Th. Gabler Verlag, Wiesbaden.

Wiese, H. (1990), *Netzeffekte und Kompatibilität: Eine theoretische und simulationsgeleitete Analyse zur Absatzpolitik für Netzeffekt-Güter*, Poeschel Verlag, Stuttgart.

Wirtschaftsministerium (2004), 'Linux & Co. – Open Source Software in der Praxis', http://www.ebigo.de/imperia/md/content/checklisten/ebigo_compact/linux [7. Sep 2004].

von Krogh, G. (2003), 'Open-Source Software Development', *Sloan Management Review* **44**(3), S. 14–18.