Jenaer Schriften zur Wirtschaftswissenschaft

Die ökonomischen Eigenschaften von Software

Sebastian v. Engelhardt

14/2006

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

Die ökonomischen Eigenschaften von Software

Sebastian v. Engelhardt
Friedrich-Schiller-Universität Jena
Wirtschaftswissenschaftliche Fakultät
Carl-Zeiß-Str. 3, D-07743 Jena

Zusammenfassung

Software ist ein Gut mit besonderen ökonomischen Eigenschaften. In diesem Artikel werden, ausgehend von einer allgemeinen Definition des Gutes Software, systematisch zentrale Eigenschaften herausgearbeitet, welche Implikationen für die Produktion und Kostenstruktur, die Nachfrage, der Bestreitbarkeit von Softwaremärkten und der Allokationseffizienz haben. Dabei hat es sich als sinnvoll erwiesen, die einzelnen Eigenschaften unter folgende Oberbegriffe zu subsummieren: Software als System zur Datenverarbeitung, Software als System von Befehlen bzw. Anweisungen, Software als rekombinierbares System, Software als ein nur in diskreten Einheiten nutzbares Gut, Software als komplexes System und Software als ein immaterielles Gut. Es zeigt sich, dass Software eine Fülle von ökonomisch relevanten Eignschaften aufweist, die von Netzwerkeffekten über subadditiver Nutzenfunktion bis hin zur Nichtrivalität reichen. Besonders hervorzuheben ist, dass Software sich von anderen Informationsgütern fundamental unterscheidet: Zum Einen fehlt ein aus Kundensicht relevanter additiver Nutzen, zum Anderen ist der durchschnittliche Nutzer/Konsument lediglich an dem Funktionieren der Algorithmen interessiert, nicht aber an der zugrundeliegenden Information.

Schlagworte: additiver Nutzen, binäre Nachfrage, digitale Güter, Erfahrungsgut, Humankapital, Informationgut, Kompatibilität, Komplexität, Netzwerkeffekte, Nichtrivalität im Konsum, Open Source, Rekombinierbarkeit, Software, subadditive Kostenfunktion, Wissen
JEL-Klassifkation: D82, D83, D62, D85, K11

Inhaltsverzeichnis

1 Der Untersuchungsgegenstand

Streng genommen bezeichnet der Begriff Software in Abgrenzung zur Hardware *alle* nichtphysischen Funktionsbestandteile eines Computers, also neben den Computerprogrammen auch die zur Verwendung mit den Programmen bestimmten Daten (vgl. Küchlin und Weber (2005), S 15 f.). Häufig wird Software jedoch in *Abgrenzung zum Begriff der Daten* mit Programmen gleichgesetzt, und so zum Beispiel als eine „Liste von Anweisungen zur Verarbeitung von Daten“ (Gröhn (1999), S. 4) definiert. Der vorliegende Text folgt dieser letztgenannten Unterscheidung und konzentriert sich dabei auf so genannte *Computersoftware*, d.h. sie ignoriert „the software embedded in electronic systems because a market does not exist for this.“ (Bitzer (1997), S. 5) Weitere Unterscheidungen, etwa in *Anwendungs- und Systemsoftware* (vgl. Gröhn (1999), S. 5), sollen hier nicht vorgenommen werden.

Software jeglicher Art liegt prinzipiell in zwei Formen vor: Einmal als sogenannter *Programmecode (Quelltext, Quellcode, Source Code)*, und einmal in der nur maschinenlesbaren Form des *Binärcodes*. Dies ergibt sich unmittelbar aus dem Programmiervorgang von Software: Eine Software, d.h. die logische Struktur, die Befehle und Anweisungen werden von Programmierern in einer sogenannten Programmiersprache geschrieben. Als Ergebnis liegt die Software als Quelltext in einer für den Menschen lesbaren und verständlichen Form vor. In einem zweiten Schritt wird dann dieser vom Menschen lesbare Quelltext in eine maschinenlesbare Form gebracht, damit das Computerprogramm vom Rechner ausgeführt werden kann. Diesen Vorgang nennt man kompilieren, und das Ergebnis ist der sogenannte Binärcode, welcher für den Menschen im semantischen Sinne praktisch nicht lesbar ist. Rückschlüsse auf die Struktur und Programmierung der Software allein auf Grund des binären Codes sind nicht oder nur sehr schwer möglich. Deshalb wird im Folgenden von der Möglichkeit eines solchen Rückschlusses vom Binärcode auf das zugrunde liegende Programmierprinzip – das sogenannte Reverse Engineering – abstrahiert, und unterstellt, dass nur bei Vorlage des Quellcodes der Ablauf und die einzelnen Programmierschritte nachvollzogen werden können (vgl. Kooths u. a. (2003), S.16 ff. sowie Hansen und Neumann, S. 155 f.).

Eine weitere Kategorie unterscheidet Softwareprodukte anhand eigentums- bzw. verwertungsrechtlicher Kriterien und/oder unterschiedlicher Produktionsweisen: die Rede ist von den Gegensatzpaaren proprietärer vs. offener Software, bzw. *Closed Source Software (CSS) vs. Open Source Software (OSS)*. Auch wenn hier weitergehende Unterscheidungen wie z.B. in nicht-kommerzielle OSS, Free-/Shareware, kommerzielle OSS und kommerzielle/propriertäre Software (vgl. BERLECON RESEARCH (2002), S. 11) möglich sind, reicht es für die folgenden Überlegungen völlig aus, auf eine stark stilisierte und typisier-

te Gegenüberstellung von OSS vs. CSS zurückzugreifen. Die Darstellung orientiert sich dabei an Pasche und v. Engelhardt (2004), S. 1:

- Bei OSS erfolgt eine Offenlegung des Source Codes und damit der zugrunde liegenden programmiertechnischen Leistung. OSS darf beliebig kopiert, verbreitet, genutzt, verändert und in der veränderten Form weitergegeben werden (vgl. detailliert die Open Source Definition in Open Source Initiative (2004)). Als kommerzielle OSS-Produkte existieren sogenannte OSS-Distributionen: Eine OSS-Distribution lässt sich allgemein als ein aufeinander abgestimmtes und optimiertes Paket von OSS beschreiben. Diesem spezifischem OSS-Paket werden in der Regel noch Leistungen wie kleinere Hilfsprogramme, Supportleistungen, Handbücher etc. hinzugefügt. Die kommerziellen OSS-Unternehmen stehen zwar mit ihren unterschiedlichen OSS-Distributionen im Wettberwerb, beteilligen sich aber durchaus mit eigenen Entwicklungsbeiträgen am selben OSS-Projekt. Dies ermöglicht ein cost sharing, vergleichbar der Situation bei Forschungskooperationen (vgl. Pasche und v. Engelhardt (2006), S. 105).
- CSS dagegen betont die Ausschließbarkeit der Nutzung von Software. D.h. im Gegensatz zur OSS wird hier der vom Menschen lesbare Quellcode nicht offengelegt und die Software wird in der lediglich maschinenlesbaren Form des Binärcodes vertrieben. Verkauft wird also ein fertiges Softwareprodukt, nicht aber das programmiertechnische Wissen, welches zu dieser Lösung führt. Der technische Vorgang des Kompilierens schützt damit die Verfügung über den Quellcode. Zudem ist auch das Kompilat, der binäre Code, mit Verfügungsrechten ausgestattet: Das Recht zur Nutzung einer Software erhält der Konsument meist durch den Erwerb einer entsprechenden Lizenz (vgl. Pasche und v. Engelhardt (2004), S. 8). Zur Durchsetzung des Ausschlussprinzips bedient man sich hier in aller Regel einer Kombination aus technischem und juristischem Kopierschutz (vgl. Kotkamp (2001), S. 53).

2 Die ökonomischen Eigenschaften von Software

Um die ökonomisch relevanten Eigenschaften des Gutes Software systematisch darzustellen, wird die nun folgende, allgemeinen Definition des Gutes Software verwendet:

> *Software ist als ein komplexes und rekombinierbares System von Befehlen bzw. Anweisungen zur Datenverarbeitung ein immaterielles und nur in diskreten Einheiten nutzbares Gut.*

Mit dieser Definition sind die einzelnen Aspekte des Gutes Software explizit benannt und es können nun die ökonomisch relevanten Eigenschaften im Einzelnen abgeleitet werden.

2.1 Software als System zur Datenverarbeitung

> *Software ist als ein komplexes und rekombinierbares **System** von Befehlen bzw. Anweisungen zur **Datenverarbeitung** ein immaterielles und nur in diskreten Einheiten nutzbares Gut.*

Das grundlegende Prinzip von Software (bzw. die grundlegende Abfolge von Vorgängen in der EDV allgemein) ist das so genannte EVA-Prinzip (vgl. Hansen und Neumann, S. 652 f.): Eingabe–Verarbeitung–Ausgabe. Immer dann, wenn Daten*verarbeitung* stattfindet, kommt es also auch zu einem Daten*austausch*, und zwar sowohl bei der Daten*eingabe* als auch bei der Daten*ausgabe*. Dies wird am Beispiel einer Textverarbeitungssoftware deutlich: Der Benutzer gibt per Tastatur Daten ein, welche vom Programm interpretiert, verarbeitet und am Bildschirm dargestellt werden. Möchte der Benutzer seinen Text ausdrucken, so gibt er den Druckbefehl. Dies wird vom Textverarbeitungsprogramm entsprechend interpretiert und die Daten werden an einen Druckertreiber gesendet. Dieser Druckertreiber (eine weitere Software) übersetzt nun die empfangenen Daten in die geeignete Druckersprache. Anschließend sendet er das Ergebnis an den Drucker, der wiederum die Daten in Form eines auf Papier gedruckten Textes ausgibt. Dieses einfache Beispiel zeigt bereits drei Formen des Datenaustausches: den Austausch zwischen Benutzer und Software, zwischen Software und Software sowie zwischen Software und Hardware. Hinzu kommt noch die Möglichkeit, dass Benutzer Daten untereinander austauschen: Wird die mit der Textverarbeitungssoftware erstellte Datei auf eine Diskette gespeichert, so kann diese Datei an einen anderen Nutzer weitergegeben und von diesem ggf. weiter bearbeitet werden. Dabei gilt, je „mehr Nutzer ein bestimmtes Betriebssystem oder eine Anwendungssoftware verwenden, desto einfacher ist es, Dateien auszutauschen“ (Kooths u. a. (2003), S. 22). Der Nutzen einer Software steigt demnach mit der Gesamtzahl (dem Netzwerk) aller Konsumenten dieser Software. Software ist eines jener „products for which the utility that a user derives from consumption of the good increases with the number of other agents consuming the good“ (Katz und Shapiro (1985) S. 424). Es treten also sogenannte Netzwerkeffekte auf, Software ist demnach ein Netzwerkgut (vgl. Pasche und v. Engelhardt (2006), S. 102). Dabei handelt es sich bei Software um ein so genanntes *virtuelles Netzwerkgut*, in Abgrenzung zu physischen Netzwerkgütern, die durch *physische* Verbindungen und Netzwerkknoten geprägt sind (vgl. Fichert (2002), S. 2).

Netzwerkeffekte lassen sich, z.B. in Anlehnung an Blankart und Knieps (1992), S. 79, allgemein durch eine Nutzenfunktion der Gestalt

$$u(x) = u_x + u(N_x) \tag{1}$$

beschreiben, mit x als dem Netzwerkgut, u_x als dem Basis- oder Stand-Alone-Nutzen des Gutes sowie $u(N_x)$ als dem Nutzen auf Grund der Netzwerkgröße N_x, wobei gilt: $\frac{du(N_x)}{dN_x} > 0$. Generell betrachtet gibt es Netzwerkgüter, deren Stand-Alone-Nutzen Null ist, beispielsweise das Telefon oder das Faxgerät.[1] Bei Software kann der Basisnutzen ebenfalls Null sein (z.B. Instant-Messanger-Programme), in der Regel kann aber von einem positiven Basisnutzen ausgegangen werden (Textverarbeitung, Tabellenkalkulation, etc.). Es sind auch Softwareprodukte vorstellbar, bei denen der Nutzen fast ausschließlich aus dem Stand-Alone-Nutzen besteht, also der Netzwerknutzen nahe Null ist (vgl. Fichert (2002), S. 4).

Bei (nachfrageseitigen) Netzwerkeffekten können Erwartungen einen großen Einfluß auf die Konsumentenentscheidung haben: Erwarten die Konsumenten, dass sich eine technologisch unterlegene Technologie b gegenüber einer Technologie a durchsetzt, dann ist b bezüglich der erwarteten Netzwerkgröße im Vorteil. Wenn der mit der erwarteten Netzwerkgröße bewertete Netzeffekt die technische Unterlegenheit ($u_a > u_b$) überkompensiert, dann wird auch tatsächlich die Technologie b gewählt (vgl. Fichert (2002), S. 5). Formal ausgedrückt:

$$E\left[u(b)\right] = u_b + E\left[u(N_b)\right] > E\left[u(a)\right] = u_a + E\left[u(N_a)\right] \tag{2}$$

bzw.

$$E\left[u(N_b)\right] - E\left[u(N_a)\right] > u_a - u_b. \tag{3}$$

Wie bereits hier deutlich wird, *kann* es bei Netzwerkgütern zu einer Dominanz inferiorer Technologien kommen.

In der Formulierung der Nutzenfunktion als $u(x) = u_x + u(N_x)$ sind die Netzwerkeffekte allgemein mit $u(N_x)$ zusammengefasst. Es lassen sich jedoch *direkte* und *indirekte* Netzwerkeffekte unterschieden (vgl. Katz und Shapiro (1985), S. 424):

Direkte Netzwerkeffekte ergeben sich *unmittelbar* aus der Netzwerk-Eigenschaft des Gutes, d.h. der Nutzen steigt mit der Verbreitung *dieses* Gutes bzw. der Verbreitung *kompatibler* Güter an (vgl. Ehrhardt (2001), S. 25). Als Beispiel wären hier die Computer-Plattformen Apple und Wintel-Standard[2] zu nennen: „Je mehr Anwender die jeweilige Plattform aufweist, desto eher kann der Einzelne problemlos Daten mit anderen austauschen“ (Ehrhardt (2001), S. 26). Die Wirkung direkter Netzwerkeffekte beschreibt ‚Metcalfe's Law‘: Ist die Anzahl der Netzteilnehmer n proportional zum Nutzen $u(.)$ des Netzwerkes bzw. des Netzwerkgutes für jeden einzelnen Teilnehmer, so gilt der Zusammenhang $u(.) \sim n \cdot (n-1)$ (vgl. Shapiro und Varian (1999), S. 184).

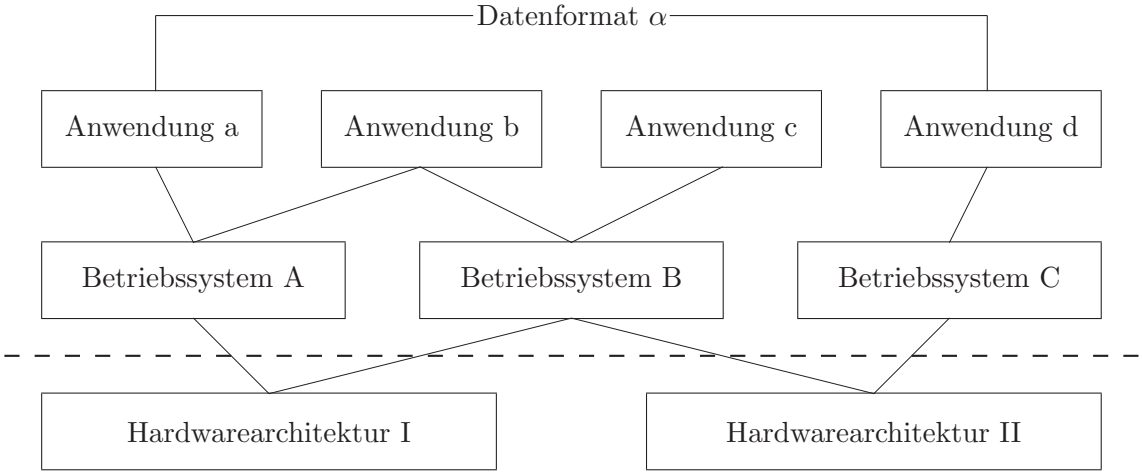
[1] Die Kopierfunktion des Faxgerätes ist hier außer Acht gelassen.

[2] Der Begriff ‚Wintel‘ ist zusammengesetzt aus ‚Windows‘ und ‚Intel‘ und bezeichnet allgemein eine PC-Architektur, die auch als ‚IBM kompatibler PC‘ bekannt ist.

Die *indirekten Netzwerkeffekte* sind im Wesentlichen auf zwei Ursachen zurückzuführen: Zum Einen auf die Bedeutung *komplementärer* Produkte und Dienstleistungen und zum Anderen auf *Lerneffekte und Information Spillovers* (vgl. Ehrhardt (2001), S. 27 ff.). Letzteres meint die Bedeutung des Wissensaustausches unter den Mitgliedern eines Netzwerkes. Je größer das Netzwerk, je mehr Nutzer dasselbe Betriebssystem oder dieselbe Anwendungssoftware verwenden, desto wahrscheinlicher ist es, relativ einfach jemanden zu finden, der die gesuchte Hilfestellung für ein Problem oder eine Antwort auf eine Frage geben kann, und umso attraktiver ist damit das Netzwerkgut (vgl. Kooths u. a. (2003), S. 22). Dieser indirekte Netzwerkeffekt der Lerneffekte und Information Spillovers ist vor allem auf dem Markt von PC-Software zu beobachten (vgl. Cowan (1992), S. 291). Bei neuen, innovativen Technologien, deren Eigenschaften und Einsatzmöglichkeiten noch relativ unbekannt sind, haben Information Spillovers noch eine weitere Wirkung: Konkurrieren neue – d.h. für die potenziellen Nutzer bezüglich der jeweilligen Effektivität noch nicht abschließend einschätzbare – Technologien miteinander, so wird diejenige Technologie für potenzielle Nutzer attraktiver sein, für die sich in der Vergangenheit bereits am meisten Nutzer entschieden haben, da hier „die Ungewissheit [bezogen auf die Leistung der Technologie] auf Grund zahlreicher vorheriger Übernahmen am geringsten ist“ (Ehrhardt (2001), S. 29). Die Bedeutung *komplementärer* Produkte und Dienstleistungen – also *komplementärer Software* – ist unmittelbar einleuchtend: ein Betriebssystem ist für die Konsumenten umso attraktiver, je mehr Anwendungsprogramme für dieses Betriebssystem zur Verfügung stehen, bzw. erwartet werden. Die nachfrageseitige Wirkung komplementärer Güter ist in der Netzwerktheorie auch als ‚Hardware-Software'-Paradigma bekannt (vgl. Katz und Shapiro (1985), S. 424). Dieser Name leitet sich aus der Tatsache ab, dass die Entscheidung für eine bestimmte Hardwarearchitektur auch implizit eine Einschränkung des Sets möglicher Betriebssysteme bedeutet (z.B. Apple vs. Wintel). Entsprechend formuliert Shy (2001): „Complementary means that consumers in these markets are shopping for *systems*“ (Shy (2001), S. 2 – Hervorhebung im Original). Darüber hinaus ist für die *Produzenten* von Anwendungssoftware die Entwicklung von Programmen für ein bestimmtes Betriebssystem umso attraktiver, je mehr Konsumenten dieses Betriebssystem benutzen, bzw. erwartet werden. Diese Bedeutung der (erwarteten) Netzwerkgröße für die Anbieter komplementärer Produkte wird auch als *angebotsseitige* Netzwerkeffekte bezeichnet (vgl. dazu z.B. Kooths u. a. (2003), S. 21). Nach Meinung des Autors lassen sich diese angebotsseitigen Netzwerkeffekte auf die nachfrageseitigen Netzwerkeffekte, auf den Systemcharakter der Güter zurückführen. Im Folgenden wird daher nicht zwischen angebotsseitiger und nachfrageseitiger Netzwerkeffekte unterschieden. Mit dem Begriff der Netzwerkeffekte sind dann immer nachfrageseitige Netzwerkeffekte gemeint.

Die Bedeutung von Komplementarität für (indirekte) Netzwerkeffekte verweist auf die Relevanz von *Kompatibilität*: Die Schaffung von Komplementarität wird in der Regel erreicht durch Kompatibilität der einzelnen Komponenten (vgl. Lopatka und Page (1999), S. 955). Kompatibilität ist die grundlegende Eigenschaft von Netzwerkgütern: Die Entscheidung darüber, ob ein Gut dem Netzwerk X (dem System X) angehört oder nicht, hängt davon ab, ob dieses Gut mit den anderen Gütern des Netzwerkes X kompatibel ist oder nicht; Kompatibilität ist daher das entscheidende Abgrenzungsmerkmal zwischen unterschiedlichen Netzwerksystemen (vgl. Katz und Shapiro (1985), S. 424 f.). Dieser Zusammenhang wird anhand von Abbildung 1 verdeutlicht, die in allgemeiner Form die Netzwerke bzw. Netzeffekte im Softwarebereich darstellt. Dabei wurden in dieser Abbildung die indirekten Netzwerkeffekte in Form von Lerneffekten und Information Spillovers außer Acht gelassen,[3] wogegen die Hardwareebene aus Gründen der Anschaulichkeit noch mit berücksichtigt wurde. Die Anwendungssoftware a lässt sich nur auf

Abbildung 1: Substitute, Komplementäre sowie horizontale und vertikale Kompatibilität



dem Betriebssystem A installieren, die Anwendung b dagegen auf A und B. Die Anwendung a und das Betriebssystem A sind demnach komplementäre Güter, sie sind vertikal kompatibel. Das Gleiche gilt für die Anwendung b und das Betriebssystem A. Bezogen auf indirekte Netzwerkeffekte sind die Betriebssysteme A und B dem Betriebssystem C überlegen, denn für diese beiden Betriebssysteme sind mehr komplementäre Anwendungen (jeweils zwei: a und b bzw. b und c) verfügbar, als für C. Die Anwendungen a und

[3]Der Grund hierfür ist, dass a) die Anzahl der Nutzer in der Abbildung 1 nicht dargestellt sind, und b) der Zusammenhang zwischen der Anzahl der Nutzer eines Netzwerkgutes und dem Auftreten *dieser* indirekten Netzwerkeffekte recht banal ist: alle Nutzer z.B. des Betriebssystems A unterliegen den Lerneffekten und Informatioaln Spillovers bezogen auf das Betriebssystem A. Hier zeigt sich eine große Nähe – oder Ähnlichkeit – zu direkten Netzwerkeffekten.

d sind – bezogen auf die direkten Netzwerkeffekte (Datenaustausch) – als Substitute zu betrachten: da beide Anwendungen mit demselben Datenformat α arbeiten, liegt hier horizontale Kompatibilität vor. Auch die Betriebssysteme A und B können als Substitute betrachtet werden, nämlich dann, wenn als Netzwerkgut die Anwendung b betrachtet wird: auf beiden Betriebssystemen kann Anwendung b installiert werden. Vergleicht man die indirekten Netzwerkeffekte der Anwendung b mit denen von Anwendung c, so ist b die überlegene Anwendung, da hier mehr Komplementärgüter – sprich: die kompatiblen Betriebssysteme A und B – zur Verfügung stehen.

2.2 Software als System von Befehlen bzw. Anweisungen

> *Software ist als ein komplexes und rekombinierbares* ***System von Befehlen bzw. Anweisungen*** *zur Datenverarbeitung ein immaterielles und nur in diskreten Einheiten nutzbares Gut.*

Der Begriff Software bezeichnet zusammenhängende Computerbefehle, also ein System von Anweisungen zur Datenverarbeitung. Jede Software kann daher als ein logisches Konstrukt von Algorithmen und Anweisungen aufgefasst werden, welches eine oder mehrere Aufgaben zu erfüllen hat. Dosi (1996) zählt zu Informationen – in Abgrenzung zum Begriff des Wissens – unter anderem auch explizit Algorithmen und Handlungsanweisungen (vgl. Dosi (1996), S. 84). Software gehört demnach zur Gruppe der Informationsgüter. Im Zusammenhang mit Informationsgütern, d.h. immer dann, wenn das Gut Information gehandelt werden soll, tritt das so genannte *Informationsparadoxon* auf (vgl. Kotkamp (2001), S. 55): Dem Käufer einer Information ist es nicht möglich, den Wert der Information vor der Transaktion zu bestimmen, es sei denn, der Verkäufer legt die gesamte Information offen. Ist die Information jedoch offen gelegt, so besteht weder Anreiz noch Notwendigkeit zum Kauf. Daher wird die Information vor dem Kauf nicht oder nur unzureichend offen gelegt, der Kauf von Information ist also immer eine Entscheidung unter Unsicherheit. Dieser Aspekt erklärt, weshalb

- kommerzielle, d.h. proprietäre Software (CSS) lediglich in Form des Binärcodes vertrieben wird: wäre der Source Code zugänglich, hätte niemand einen Anreiz diese Software käuflich zu erwerben. Entsprechend existiert im OSS-Bereich auch kein Markt für die OSS an sich, sondern nur für komplementäre Güter und Dienstleistungen. Allerdings werden diese häufig im Paket mit der OSS zusammen als sogenannte OSS-Distribution vertrieben (vgl. auch Abschnitt 1).
- Konsumenten sowie die Öffentlichkeit allgemein den CSS -Produzenten häufig mit einem gewissen Misstrauen begegnen, da wegen der fehlenden Offenlegung des Co-

des auch *nach* dem Kauf die Informationen (die Algorithmen) dem Konsumenten nicht zugänglich sind. So entspann sich ab Dezember 1999 eine heftige Diskussion über das im Betriebssystem Windows 2000 enthaltene Defragmentierungsprogramm DisKeeper. Die Computerzeitschrift c't hatte berichtet, dass DisKeeper von einer Firma Namens ‚Executive Software Inc.' stammt, dessen Gründer und Executive-Chef Craig Jensen ein bekennender Scientologe ist (vgl. Göhring (1999), S. 58 ff.). Da ein solches Programm prinzipiell „direkten und aktiven Zugriff auf alle Daten besitzt" (Göhring (1999), S. 58), führte der Bericht zu deutlichen Reaktionen: So riefen Sektenbeauftragte zum Windows-2000-Boykott auf (vgl. Heise News (2000)), katholische Bistümer rieten ihren Pfarreien von einem Einsatz des Betriebssystems Windows 2000 ab (vgl. Spiegel Online (2000)) und das Bundesamt für Sicherheit in der Informationstechnik forderte, Einsicht in den Quellcode nehmen zu dürfen, um mögliche Gefahren durch die Defragmentier-Software abschätzen zu können[4]. Auch die Diskussion, ob der Zweitschlüssel ‚NSAKey' in der Windows Crypto API unter der Kontrolle des US-Geheimdienstes NSA stehe (vgl. Heise News (1999)) und die Frage nach einer von den Nutzern in der Regel nicht bemerkten Datensammlung durch Micorsoft (vgl. z.B. den kritischen Bericht zur Übertragung einer System ID an Microsoft von Persson und Siering (1999), S. 16 ff.) sind Ausdruck einer Verunsicherung der Konsumenten. Unabhängig davon, ob solche Vorwürfe und Verdächtigungen im Einzelfall zutreffen oder nicht, sind sie Folge einer auch nach dem Kauf noch bestehenden Unsicherheit der Konsumenten, da die Information (der Algorithmus) hier weiterhin verborgen bleibt.

Es fällt auf, dass das Gut Software in zwei Varianten auftritt: Entweder ist die *Information von Anfang an frei zugänglich* – dann handelt es sich um *nicht am Markt gehandelte OSS*, oder aber es liegt *am Markt gehandelte CSS* vor – in diesem Fall bleibt die *Information jedoch vor und nach dem Kauf verborgen.* Damit unterscheidet sich Software deutlich von anderen Informationsgütern. Nach dem Kauf beispielsweise einer Tageszeitung sind die darin enthaltenen Informationen dem Konsumenten selbstverständlich zugänglich. Der Grund für den Unterschied besteht darin, dass es bei Software auf der Informationsebene keinen für den *Konsumenten relevanten* ‚additiven Nutzen' gibt. Mit dem Begriff des ‚additiven Nutzens' ist Folgendes gemeint: Der Nutzen einer Tageszeitung lässt sich gedanklich unterteilen in den Nutzen der darin enthaltenen Information und den additiven Nutzen der Auswahl und Aufbereitung dieser Information (der redaktionellen oder journalistischen Leistung). Die Existenz und Relevanz des additiven Nutzens sind ein ent-

[4]Dies wurde allerdings von Microsoft verweigert. Die europaweit bisher erste Regierungsstelle, die Einblick in den Sourcecode erhielt, ist das österreichische Bundesministerium für Inneres (vgl. Heise News (2001)).

scheidendes Alleinstellungsmerkmal für Tageszeitungen: so unterscheiden sich die großen überregionalen Tageszeitungen hauptsächlich in der unterschiedlichen Gewichtung und Aufbereitung, also dem unterschiedlichen additiven Nutzen. Die Bedeutung des additiven Nutzens kann auch am Beispiel wissenschaftlicher Texte deutlich gemacht werden: zwei Aufsätze zu demselben Thema oder derselben These beinhalten vielleicht exakt die gleiche Information, können jedoch auf Grund ihres unterschiedlichen additiven Nutzens vom Konsumenten als qualitativ sehr verschieden beurteilt werden. So wird auch verständlich, warum das Urheberrecht vom Ansatz her die additive Leistung schützt und nicht die zu Grunde liegende Information. Bei einem wissenschaftlichen Aufsatz ist durch das Urheberrecht die konkrete Ausformulierung und nicht die zu Grunde liegende Erkenntnis geschützt (vgl. Quah (2003), S. 24).

Bei Software hingegen fehlt ein solcher für den Konsumenten relevanter, additiver Nutzen: Eventuell vorhandene additive Aspekte wie eine elegante Programmierung, eine gute Dokumentation und hinreichende Kommentierung des Quellcodes interessieren lediglich die Programmierer (also die Produzenten) und nicht die Nutzer (also die Konsumenten). Für den Nutzen des Informationsgutes Software ist das ‚Funktionieren' des Algorithmus, der programmiertechnischen Lösung maßgeblich und entscheidend, während ein eventueller additiver Nutzen für das *Funktionieren* der Software keine Rolle spielt. Entsprechend sind Softwarekonsumenten auch primär an der *Wirkung* der Algorithmen und nicht am Algorithmus selbst interessiert. Hinzu kommt, dass z.B. der ausgedruckte Quellcode einer Software für die meisten Nutzer wertlos ist, da ihnen das Wissen fehlt, diesen Quellcode auch zu verstehen. Entscheidend ist für die Konsumenten also die Tatsache, dass der Algorithmus einer Software ein *Ergebnis* produziert, d.h. die *Wirkung* von Software interessiert: ein Film wird abgespielt, eine pdf-Datei erscheint auf dem Bildschirm etc. Dies erklärt, warum das Informationsgut Software auch dann für den Konsumenten Nutzen stiftet, wenn die Information nach dem Kauf nicht offen gelegt wird.

Wie bei allen Informationsgütern treten auch bei der Produktion von Software hohe Entwicklungskosten, so genannte First Copy Costs, auf, die außerdem als versunkene Kosten (Sunk Costs) zu betrachten sind (vgl. Kooths u. a. (2003), S. 20). Demgegenüber sind die Kosten der Reproduktion der fertigen, in digitaler Form vorliegenden, Software eher gering. Die entscheidenden Produktionskosten sind also die versunkenen Kosten für das Design, die Programmierung, die Testdurchläufe etc., wohingegen die Kosten für den anschließenden Kopiervorgang, die dabei verwendeten Speichermedien, die Verpackung und den Vertrieb zu vernachlässigen sind (siehe auch Abschnitt 2.6) (vgl. Pasche und v. Engelhardt (2004), S. 5 f.). Diese First Copy Costs sind eine Funktion der Komplexität der Software (Bitzer (1997), S. 7). So verursachte das Betriebssystem Windows 3.1

Entwicklungskosten in Höhe von ca. 50 Millionen US-Dollar, während bei Windows 2000 die Entwicklungskosten etwa eine Milliarde US-Dollar betrugen (vgl. Stelzer (2000), S. 837 bzw. Arthur (1996), S. 103 sowie operating-system.org (2004)).

Ist der Source Code einer Software allgemein – oder einer bestimmten Gruppe – zugänglich, so treten Spillovereffekte auf, die als Wissens-Spillover interpretiert werden können (vgl. Pasche und v. Engelhardt (2004), S. 18). Da der Terminus ‚Wissens-Spillover' den Begriff Wissen verwendet, soll an dieser Stelle zunächst kurz auf die Unterscheidung der Begriffe ‚Information' und ‚Wissen' eingangen werden: Laut der in diesem Text verwendeten Definition von Dosi (1996) sind Algorithmen Information. Daher wird der Source Code einer Software hier als Informationsgut betrachtet. „Wissen hingegen beinhaltet kognitive Kategorien, Regeln zur Interpretation von Informationen, Tacit-Fähigkeiten und Problemlösungs- und Suchheuristiken, die sich nicht in exakten Algorithmen abbilden lassen" (Kotkamp (2001), S. 30). Zur Erklärung dieser Abgrenzung wählt Dosi das Beispiel eines mathematischen Beweises: Der Beweis an sich ist Information, doch nur wenige Mathematiker haben das Wissen, diesen Beweis auch zu verstehen (vgl. Dosi (1996), S. 84). Ist dagegen im Folgenden von Wissens-Spillover die Rede, so meint dies die Diffussion der Information *und* die sich daraus ergebenen Wirkungen auf das Wissen einer Population: Wird ein mathematischer Beweis z.B. in einer Fachzeitschrift veröffentlicht, so hat dies eine Reihe von Folgen, die über den primären Effekt der Verbreitung der Information an sich hinausgehen. Manche Mathematiker werden durch das Studium des Beweises ihre mathematischen Fähigkeiten weiterentwickeln, einige Wissenschaftler nutzen die im Beweis enthaltenen logischen Schritte und Erkenntnisse zur Lösung anderer Probleme usw. *Alle* diese Effekte einer Veröffentlichung, d.h. die Wirkung z.B. auf die mathematischen Fähigkeiten wie auch die Diffusion der Information an sich, werden also im Folgenden unter dem Begriff der Wissens-Spillover subsumiert.[5] Analog zum Beispiel aus der Mathematik treten auch bei einer Veröffentlichung von Quellcodes Spillovereffekte auf. Die Wirkung der Wissens-Spillover ist dabei sehr breit gefächert: Neben der reinen Informationsdiffusion wird eine generelle Verbreitung und Verbesserung der Programmierfähigkeiten in der betrachteten Population auftreten, ferner können Teile des Programmcodes für die Entwicklung neuer Software verwendet, d.h. direkt übernommen und integriert werden. Dieser zuletzt genannte Spillovereffekt verweist auf eine Eigenschaft von Software, die das Thema des nächsten Abschnitts ist: die Rekombinierbarkeit.

[5]Siehe zur Interdependenz von Wissen und Information auch Quah (2003), S. 36 f. sowie Loasby (1999), S.148).

2.3 Software als rekombinierbares System

Software ist als ein komplexes und ***rekombinierbares*** *System von Befehlen bzw. Anweisungen zur Datenverarbeitung ein immaterielles und nur in diskreten Einheiten nutzbares Gut.*

Laut Quah (2003) zählt Software zu den *digitalen Gütern* (vgl. Quah (2003), S. 29). Entsprechend trifft folgende Aussage auch auf das in der vorliegenden Arbeit betrachtete Gut zu: „Digital goods are recombinant. By this I mean they are cumulative and emergent new digital goods that arise from merging antecedents have features absent from the original, parent digital goods“ (Quah (2003), S. 19). Wie im Abschnitt 2.2 angedeutet, können bereits geschriebenen Programmelemente in andere, d.h. neue Programme eingesetzt werden. Aus diesem Grund existieren auch ganze Bibliotheken aus fertigen Programmelementen (vgl. Gröhn (1999), S. 5) und es hat sich ein eigener Programmieransatz – das so genannte komponentenbasierte Softwareengineering – entwickelt, welcher auch eine herstellerübergreifende Wiederverwendung von Programmkomponenten umfasst (vgl. dazu Romberg (2003), S. 253 ff.). Rein theoretisch besteht also die Möglichkeit, eine neue Software vollständig aus Teilstücken vorhandener Softwareprodukte zu erstellen – zu *rekombinieren* – ohne dass eine einzige neue Zeile Quellcode geschrieben werden muss. In der Praxis werden jedoch *vorhandene* Programmteile mit *neuem* Quellcode kombiniert.[6] Auch die ständige Weiterentwicklung (Versionen) von Software verdeutlicht die Rekombinierbarkeit: Aus einer alten Version wird durch die Rekombination des alten Quellcodes mit neuen Quellcode-Elementen eine neue Version, eine neues Softwaregut. Diese Rekombinierbarkeit kann als ein lokaler Spillovereffekt verstanden werden. Oder anders ausgedrückt: bei der Softwareproduktion ist das Auftreten von *Verbundvorteilen* sehr wahrscheinlich. Verbundvorteile ergeben sich aus der Möglichkeit, bei der Produktion von n Gütern Einsatzfaktoren zu nutzen, die Kollektivgutcharakter[7] haben (vgl. Windisch (1987), S. 50). Auf Software bezogen bedeutet dies: die Verbundvorteile zeigen sich darin, dass dieselben Quellcode-Bausteine für die Produktion verschiedener, also für

[6] Ein populäres Beispiel hierfür ist die Programmierumgebung Delphi der Firma Borland: Vereinfacht ausgedrückt besteht die Programmierarbeit unter Delphi daraus, vordefinierten Komponenten Eigenschaften zuzuweisen und zu kombinieren. Es ist zudem möglich eigene Komponenten zu entwickeln (vgl. Bohne und Lang (2000)).

[7] Der Begriff des Kollektivgutes bzw. Kollektivgutcharakters stellt hier auf die Eigenschaft ab, „dass jene Faktoren von mehreren Produktlinien gemeinsam genutzt werden können, ohne dass die Nutzung durch eine Produktlinie die durch eine andere beeinträchtigt“ (Windisch (1987), S. 50). Als Beispiel führt Windisch (1987) hier unter anderem nicht ausgelastete Maschinen an. Die Begrifflichkeit ist hier beibehalten worden, auch um eine *sprachliche* Abgrenzung zur Frage der grundsätzlichen Bereitstellung von Software – d.h. als Clubgut oder als öffentliches Gut – zu erhalten, wohl wissend, dass die Begriffe Kollektivgut und öffentliches Gut letztlich Substitute sind.

die Produktion von n Softwaregütern verwendet werden können, ohne dass es zu einer Rivalität in der Nutzung der Quellcode-Elemente kommt. Die Quellcode-Bausteine stellen daher Einsatzfaktoren dar, die Kollektivgutcharakter haben.

2.4 Software als ein nur in diskreten Einheiten nutzbares Gut

> *Software ist als ein komplexes und rekombinierbares System von Befehlen bzw. Anweisungen zur Datenverarbeitung ein immaterielles und nur in* ***diskreten Einheiten nutzbares*** *Gut.*

Software ist ein nur in diskreten Einheiten nutzbares Gut (vgl. Quah (2003), S. 17). Das bedeutet zum Einen, dass Unteilbarkeit vorliegt: Eine nur zu 50% überspielte Software ist ‚gar keine Software', während beispielsweise 50% eines Apfels ein halber Apfel und nicht kein Apfel sind. Software ist also nur in ganzzahligen Mengen nützlich.

Zum Anderen wird Software – grundlegend betrachtet – auch immer nur in der Menge *Eins* konsumiert: pro Kombinationseinheit Konsument-Computer wird eine bestimmte Software sinnvoller Weise nur einmal installiert, es macht keinen Sinn die gleiche Software mehrfach zu installieren. Selbstverständlich können mehrere Konsumenten ein und denselben Computer benutzen, und deshalb auch so genannte Paralellinstalation vorliegen. Allerdings handelt es sich hierbei um mehrere Kombinationseinheiten Konsument-Computer. Grundsätzlich gilt also, dass der Konsum einer beliebigen Software a pro Konsument-Computer Kombination immer eine binäre Größe darstellt:

$$c(a) = x, \qquad \text{mit } x \in [0,1], \in IN \tag{4}$$

Die Gesamtnachfrage $\sum_{m=1}^{n} c_m(a) = C(a)$ von n Kombinationseinheiten Konsument-Computer

$$C(a) = X \qquad \text{mit } X \in IN. \tag{5}$$

ist demnach eine *diskrete* Funktion.

Die Tatsache, dass Software nur in diskreten Einheiten nutzbar ist, bzw. dass bei Software Unteilbarkeit vorliegt, weist auf die sogenannte ‚Zerbrechlichkeit' von Software hin (vgl. Quah (2003), S. 17). Diese Zerbrechlichkeit liegt in der Immaterialität und der Komplexität von Software gleichermaßen begründet: Eine nur zu 99% korrekt überspielte Software ist unbrauchbar, da Software aus einem komplexen System von Befehlen besteht (im übertragenen Sinne eine logische Maschine darstellt), welches *dann* nicht mehr funktioniert, wenn an einer, auch noch so kleinen Stelle, die *entscheidende* Befehlskette fehlt. Software als komplexes System ist das Thema des nächsten Abschnitts.

2.5 Software als komplexes System

> *Software ist als ein **komplexes** und rekombinierbares **System von Befehlen bzw. Anweisungen** zur Datenverarbeitung ein immaterielles und nur in diskreten Einheiten nutzbares Gut.*

Software ist ein sehr komplexes Gebilde, aufgebaut aus Wenn-Dann-Abfolgen, logischen Schleifen, Entweder-Oder-Schaltungen usw. Auf Grund der Komplexität ist es praktisch *unmöglich* eine völlig *fehlerfreie* Software zu programmieren (vgl. Computer Zeitung (2003)) – zumindest, wenn man entsprechend umfangreiche Software betrachtet, also von Trivialfällen wie z.B. einem sehr simplen Texteditor etc. absieht. So geht man beispielsweise davon aus, dass proprietäre Software im Durchschnitt 0,51 Fehler pro Tausend Zeilen Programmcode aufweist (vgl. tecChannel News (2003)). Da das Betriebssystem Windows 2000 aus schätzungsweise 30 Millionen Zeilen besteht (vgl. Hochschulrechenzentrum (2003)), bedeutet dies, dass hier mit rund 15.000 Fehlern zu rechnen ist. Einige dieser Fehler treten nie zu Tage, andere äußern sich in Programm- oder Systemabstürzen. Die Tatsache, dass Software sicherheitsrelevante Lücken aufweist und so eine Angriffsfläche für Viren bietet, lässt sich als eine besondere Form von Softwarefehler interpretieren.

Bei den Fehlern in Software werden Programmierfehler, Anwenderfehler und Systemfehler unterschieden. Programmierfehler können als im weitesten Sinne logische Fehler und Flüchtigkeitsfehler auf der Ebene des Programms, der internen Programmlogik umschrieben werden (vgl. Bohne und Lang (2000), S. 460 ff.). Beispiele hierfür sind neben Tippfehlern so genannte Endlosschleifen und zyklische Abrufe, Überlauffehler oder auch eine fehlende Freigabe von Ressourcen. Dagegen treten Anwenderfehler (Benutzerfehler) auf, wenn der Benutzer das Programm ‚falsch' benutzt, ungültige Eingaben macht, Handlungen in der falschen Reihenfolge ausführt oder etwas Unvorhergesehenes mit dem Programm macht (vgl. Bohne und Lang (2000), S. 477). Diese Fehler zählen zu den Fehlern in der Programmierung, da Programme so geschrieben sein sollten, dass die Anwender soweit wie nötig geführt werden (vgl. Bohne und Lang (2000), S. 477), also zum Beispiel fehlerhafte Eingaben nicht erlaubt sind. Dafür gibt es im Alltag positive Beispiele: Wer bei der Online-Auskunft der deutschen Bahn einen nicht eindeutigen Ziel-Namen eingibt, oder das Datum in einem unsinnigen Format, erhält eine entsprechende Rückmeldung. Das System verhindert hier eine ungültige Eingabe. Die letzte Kategorie – die der Systemfehler – betrifft die Interaktion der Software mit der Umgebung. Solche Systemfehler treten beispielsweise auf bei einem Mangel an ausreichenden Ressourcen (z.B. Arbeitsspeicher) oder bei nicht zugriffsbereiten Peripheriegeräten (z.B. ein nicht eingeschalteter Drucker) (vgl. Bohne und Lang (2000), S. 482).

Die volkswirtschaftlichen Kosten fehlerhafter Software in den USA werden auf bis zu einem Prozent des US-amerikanischen Bruttoinlandsproduktes geschätzt (vgl. NIST und Technology (2002), S. ES-2). Laut einer Studie des US-amerikanischen National Institute of Standards and Technology verursachen unzureichende Qualitätskontrollen in der Softwareentwicklung jährlich Kosten für die US-Wirtschaft in Höhe von rund 60 Millionen Dollar. Die Studie, welche sich auf Softwareentwickler und Anwender in der Automobil- und Luftfahrtindustrie sowie dem Finanzsektor konzentriert, kommt zudem zu dem Schluss, dass mit einer verbesserten Qualitätsprüfung eine Kostenreduktion in Höhe von rund 22 Millionen Dollar erreicht werden könnte (vgl. NIST und Technology (2002), S. ES-11, 1 ff. sowie Heise News (2002)). Auch die von Viren verursachten Schäden sind nicht zu vernachlässigen. So beziffert der US-amerikanische IT-Verband Computer and Communications Industry Association die Schäden allein des SoBig Wurmes auf $ 30 Milliarden und verweist auf die Angaben einer Londoner Firma Namens mi2g Ltd., wonach „global damage from malicious software inflicted as much as $107 billion in global economic damage“ (Bace u. a. (2003), S. 10).

Die Komplexität und Fehlerhaftigkeit von Software bedeutet, dass ein Großteil der Arbeitsleistung beim Programmieren aus Fehlersuche und Fehlerbeseitigung (Debugging) besteht. Ein Teil der Fehler tritt erst während der Benutzung der Software durch die Konsumenten zu Tage. Für die Nutzung von Software ist dauerhafte Wartung und Pflege (Maintenance) notwendig (vgl. Franck und Jungwirth (2002), S. 125). Laut Raymond (1999) entfallen über 75 % der Kosten einer Software in einem Unternehmen auf Maintenance. Neben der konkret-individuellen Wartung und Pflege als eine *komplementäre* Dienstleistung durch den Produzenten der Software oder durch dritte Anbieter spielt für Softwarenutzer die Partizipation an der laufenden Weiterentwicklung bzw. Verbesserung des Softwareproduktes in Form allgemein zugänglicher Updates und Patches eine große Rolle. Letzteres geschieht hauptsächlich über Internet-Downloads. Insofern lässt sich behaupten, dass Software ein Gut darstellt, dass in einem *nicht-endgültigen Zustand* ausgeliefert wird, und durch komplementäre Dienstleistung und/oder Updates und Patches laufend angepasst und/oder verbessert wird.

Die Tatsache, dass einige Fehler erst bei der Benutzung der Software erkannt werden, weist auf eine weitere Eigenart von Software hin, die sich aus der Eigenschaft der Komplexität ableiten lässt: Software ist ein so genanntes *Erfahrungsgut* (vgl. Nelson (1970)), und zwar im *doppelten* Sinne (vgl. Pasche und v. Engelhardt (2004), S. 3 sowie Pasche und v. Engelhardt (2006), S. 101 f.):

Ein Erfahrungsgut im herkömmlichen Sinne (vgl. Ewers u. a. (2003), S. 284 f.) ist Software aus folgendem Grund: Die Eigenschaften (und damit auch evtl. vorhandene Fehler) und Problemlösungsfähigkeiten des Softwareproduktes können vom Konsumenten 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* (vgl. Pasche und v. Engelhardt (2004), S. 3). Um diese Unsicherheit abzubauen, bieten Softwarehersteller häufig auf ihrer Homepage eine 30-Tage-Testversion zum Download an.

Hinzu kommt, dass der Nutzer durch den täglichen Umgang mit der Software *spezifisches Humankapital* aufbaut, der Nutzer also zum *erfahrenen* Nutzer wird. Dies ist die *zweite* Bedeutung von Software als Erfahrungsgut (vgl. Pasche und v. Engelhardt (2004), S. 3): Durch den täglichen Umgang mit der Software *lernt* der Nutzer diese immer besser kennen und erweitert seine Fähigkeiten, er hat dann ‚Erfahrung im Umgang mit der Software XY' oder ‚Erfahrung im Umgang mit gängigen Office-Programmen'. Die Produktivität im Umgang mit der Software steigt mit dem so generierten Humankapital. Für einen Nutzer der völlig ohne Erfahrungen oder Wissen ist, kann Software sogar nutzlos sein. Dies macht deutlich, dass der Nutzen im Zusammenspiel des Softwareproduktes mit dem Humankapital generiert wird – ein illustratives Beispiel für Beckers Theorie der Haushaltsproduktion (vgl. Becker und Michael (1973)). Dieses Humankapital kann in Form einer spezifischen oder einer eher allgemeinen Kompetenz aufgebaut werden, d.h. am einen Ende der Skala steht das auf eine bestimmte hochspezielle Software gerichtete Wissen, während sich am anderen Ende die generelle Fähigkeit zum Umgang mit Computern (mit verschiedenen Betriebssystemen) befindet. Software unterscheidet sich dabei im Grunde nicht von anderen Werkzeugen oder Maschinen, auch hier wird durch den täglichen Umgang – durch die Erfahrung – Humankapital aufgebaut (vgl. Arrow (1962)), welches eher allgemeiner oder spezifischer Natur sein kann. Dasselbe gilt für die Tatsache, dass Werkzeuge nur dann effizient eingesetzt werden können, wenn die Nutzer über entsprechendes Wissen verfügen, dies gilt umso mehr, je komplizierter und komplexer das Werkzeug bzw. die Maschine ist.

Das software-spezifische Humankapital ist eine Ressource (oder Kompetenz), welche sich auf dem Arbeitsmarkt verwerten lässt. Die Bedeutung zeigt sich z.B. bei Stellenangeboten, wo entsprechende Anforderungen gestellt werden, aber auch bei Stellengesuchen, in denen auf solche Kompetenzen verwiesen wird. Auch 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 (vgl. Pasche und v. Engelhardt (2004), S. 3).

Zusammenfassend lässt sich also sagen:

- Der potenzielle Nutzen einer Software kann oft erst nach dem Kauf vollständig

festgestellt werden (Erfahrungsgut im ersten Sinne),

- der realisierte Nutzen aus der Softwareverwendung ergibt sich erst durch das Zusammenspiel der Software mit entsprechendem Humankapital,[8]
- wobei Letzteres während der Verwendung der Software aufgebaut wird (Erfahrungsgut im zweiten Sinne),
- und eine am Arbeitsmarkt verwertbare Ressource oder Kompetenz darstellt.

2.6 Software als ein immaterielles Gut

Software ist als ein komplexes und rekombinierbares System von Befehlen bzw. Anweisungen zur Datenverarbeitung ein ***immaterielles*** *und nur in diskreten Einheiten nutzbares Gut.*

Software ist *immateriell*, oder wie Quah (2003) es ausdrückt: Software ist nicht-räumlich (vgl. Quah (2003), S. 18). Software benötigt zwar ein Trägermaterial, ein Speichermedium, um zu existieren (vgl. Cowan und Harison (2001), S. 2), ist aber nicht mit dem Speichermedium gleichzusetzen. So ist beispielsweise eine ‚SuSE Linux CD' oder eine ‚Windows XP CD' lediglich eine CD, auf der die Software ‚SuSE Linux' bzw. ‚Windows XP' gespeichert ist. Abstrakter formuliert: Die CD-ROM ist das *Trägermaterial*, *auf dem* das immaterielle Gut, kodiert als eine Abfolge von Nullen und Einsen, gespeichert ist.

Diese Abfolge von Nullen und Einsen kann beliebig oft dupliziert, d.h. kopiert werden und das Ergebnis – die Kopie – ist *ein zweites Original*. Dies lässt sich vergleichen mit der Kopierbarkeit von einem in Buchstaben kodierten Text: auch dieser kann dupliziert werden und das Ergebnis ist ebenfalls ein zweites Original, mag sich auch das Trägermaterial oder Speichermedium geändert haben. So liegt beispielsweise bei einem Text aus einem Buch und einer Fotokopie dieses Textes der Text einmal in gebundenen Form vor (Buch), und einmal als lose DIN A4 Blätter (Kopie), dies ändert jedoch nicht den *Inhalt* des Textes. Analog verhält es sich mit der Reproduzierbarkeit von Software, allerdings wird Software digital kopiert, d.h. ein Qualitätsverlust findet – anders als im analogen Kopiervorgang – nicht statt. Die *Kosten* der Reproduktion von Software sind dabei *nahe Null*, und lediglich durch die Kosten des verwendeten Trägermaterials (z.B. DVD) sowie den Verschleiß der physikalischen Kopiertechnologie (z.B. DVD-Brenner) bestimmt. Diese Eigenschaft wird als *unbeschränkte Reproduzierbarkeit* bezeichnet (vgl. Quah (2003), S. 13). Eine einmal erstellte Software stellt also eine *nicht knappe Ressource* dar, lediglich das Trägermaterial (z.B. CD-ROM) kann Knappheitsrestriktionen unterliegen.

[8] Der gleiche Gedanke, nur allgemein auf Information bezogen, findet sich bei Kotkamp 2001, S. 56.

Ein Gut, welches unbeschränkt reproduzierbar ist, kann auch keine Rivalität im Konsum aufweisen (vgl. Quah (2003), S. 15).[9] Diese *Nichtrivalität im Konsum* von Software gilt ebenso in zeitlicher wie mengenmäßiger Perspektive. Da sich das immaterielle Gut Software im Gebrauch nicht abnutzt, also keinem physischen Verschleiß oder Verbrauch unterliegt (vgl. Fichert (2002), S. 3 sowie Kooths u. a. (2003), S. 21) können beliebig viele Individuen dieselbe Software *nacheinander* nutzen. Aber auch mengenmäßig treten keine Restriktionen auf, da eine einmal erstellte Software beliebig oft zu Grenzkosten von (fast) Null duplizierbar ist und damit beliebig vielen Individuen zum zeitgleichen Konsum bereit gestellt werden kann.

3 Abschließende Kurz-Darstellung

Die Erkenntnisse des vorangegangenen Abschnitts lassen sich wie folgt zusammenfassen: Software weist als Netzwerkgut sowohl direkte als auch indirekte (Informational Spillover, Komplementärgüter) Netzwerkeffekte auf. Daraus folgt: Eine Software ist tendenziell um so attraktiver, je mehr Nutzer dieses Produkt verwenden. Software zählt zu den Informationsgütern, unterscheidet sich aber z.B. von Tageszeitungen dadurch, dass es keinen aus Kundensicht relevanten additiven Nutzen gibt. Zudem sind Softwarenutzer in der Regel nicht an der zugrundeliegenden Information (Algorithmus) interessiert, sondern an dem Funktionieren der Algorithmen. Dies in Verbindung mit dem Informationsparadoxon erklärt die Zweiteilung der Sofware in Closed- und Open-Source-Software. Bei Software treten hohe Entwicklungskosten, sogenannte First-Copy-Costs auf. Bei einer Offenlegung des Quellcodes ist mit Wissensspillovern zu rechnen. Die Rekombinierbarkeit von Software führt zu Verbundvorteilen in der Produktion. Software ist ein nur in diskreten Einheiten nutzbares Gut, und die individuelle Nachfrage nach einem Softwareprodukt ist als binär zu charakterisieren. In gewisser Weise lässt sich sagen, dass Software in einem nicht-endgültigen Zustand ausgeliefert wird: auf Grund der Komplexität ist es unmöglich, eine völlig fehlerfreie Software zu programmieren, zumal sich ein Teil der Fehler erst in der konkreten Anwendung zeigt. Software ist ein Erfahrungsgut im doppelten Sinne: die Eignung/Qualität eines Softwareproduktes lässt sich teilweise erst nach dem Kauf, bei der täglichen Anwendung, beurteilen. Außerdem baut der Benutzer beim Gebrauch der Software spezifisches Humankapital auf. Ein fertig programmiertes Softwareprodukt ist praktisch unbeschränkt reproduzierbar und stellt somit eine nicht-knappe Ressource dar, was auch eine Nicht-Rivalität im Konsum bedeutet.

[9] Dieser Zusammenhang gilt immer. Umgekehrt ist dies jedoch nicht zwingend, so gibt es Güter, welche zwar eine Nichtrivalität im Konsum aber trotzdem *keine* unbeschränkte Reproduzierbarkeit aufweisen (vgl. dazu auch Quah (2003), S. 16).

Literatur

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

[Arthur 1996] Arthur, W. B.: Increasing Returns and the New World of Business. In: *Harvard Business Review* (1996), S. 100–109

[Bace u. a. 2003] Bace, Rebecca ; Geer, Daniel ; Gutmann, Peter ; Metzger, Perry ; Pfleeger, Charles P. ; Quarterman, John S. ; Schneier, Bruce: CyberInsecurity: The Cost of Monopoly. How the Dominance of Microsoft's Products Poses a Risk to Security / Computer & Communications Industry Association. Washington, 2003. – Report. www.ccianet.org/papers/cyberinsecurity.pdf

[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

[BERLECON RESEARCH 2002] BERLECON RESEARCH, GmbH (Hrsg.): *FLOSS Final Report. Basics of Open Source Software Markets and Business Models.* Berlin, 2002. (Free/Libre and Open Source Software: Survey and Study 3). – www.infonomics.nl/FLOSS/report/reportPart3_basics_oss_markets_and_business_models.htm

[Bitzer 1997] Bitzer, Jürgen: The Computer Software Industry in East and West. Do Eastern European Countries Need a Specific Science and Technology Policy? / Deutsches Institut für Wirtschaftsforschung. Berlin, 1997 (149). – DIW Diskussionspapiere

[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)

[Bohne und Lang 2000] Bohne, Andreas ; Lang, Guido: *Go To Delphi 5.* München [u.a.] : Addison-Wesley-Longman, 2000

[Computer Zeitung 2003] Computer Zeitung: *Pragmatischer Umgang mit Softwarefehlern empfohlen. – Nr. 18/2003, S. 11.* 2003. – www.xcc.de/upload/pressespiegel/17_Computer_Zeitung.pdf

[Cowan 1992] Cowan, Robin: High Technology and the Economics of Standardization. In: Dierkes, Meinolf (Hrsg.): *New Technology At the Outset: Social Forces in the Shaping of Technological Innovations.* Frankfurt am Main : Campus Verlag, 1992, S. 279–300

[Cowan und Harison 2001] Cowan, Robin ; Harison, Elad: Protecting the Digital Endeavour. Prospects For Intellectual Property Rights in the Information Society / MERIT, Maastricht Economic Research Institute on Innovation and Technology. Maastricht, 2001 (28). – Research Memoranda

[Dosi 1996] Dosi, Giovanni: The Contribution of Economic Theory to the Understanding of a Knowledge-Based Economy. In: *Employment and Growth in the Knowledge-Based Economy.* Paris, 1996 (OECD Documents), S. 81–92

[Ehrhardt 2001] Ehrhardt, Marcus: *Netzwerkeffekte, Standardisierung und Wettbewerbsstrategie.* Wiesbaden : Deutscher Universitäts-Verlag, 2001 (Gabler Edition Wissenschaft: Strategische Unternehmungsführung)

[Ewers u. a. 2003] Ewers, Hans-Jürgen ; Fritsch, Michael ; Wein, Thomas: *Marktversagen und Wirtschaftspolitik. Mikroökonomische Grundlagen staatlichen Handelns.* München : Vahlen, 2003

[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

[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

[Göhring 1999] Göhring, Hans-Peter: Windows 2000 droht ein Bann. Kritik aus den Kirchen an Scientology-Beteiligung. In: *c't - Magazin für Computertechnik* 25 (1999), S. 58–61

[Gröhn 1999] Gröhn, Andreas: *Netzwerkeffekte und Wettbewerbspolitik. Eine ökonomische Analyse des Softwaremarktes.* Tübingen : Mohr Siebeck, 1999

[Hansen und Neumann] Hansen, Hans R. ; Neumann, Gustaf: *Grundlagen betrieblicher Informationsverarbeitung.* Stuttgart : Lucius & Lucius

[Heise News 1999] Heise News: *Debatte um NSAKey geht weiter.* Online Artikel. Sep. 1999. – www.heise.de/newsticker/meldung/6019

[Heise News 2000] Heise News: *Sektenbeauftragter ruft zum Windows-2000-Boykott auf.* Online Artikel. Jan. 2000. – www.heise.de/newsticker/meldung/7708

[Heise News 2001] Heise News: *Wien darf Windows-Quellcode zuerst prüfen.* Online Artikel. Dez. 2001. – www.heise.de/newsticker/meldung/23105

[Heise News 2002] Heise News: *Bugs kosten fast sechzig Milliarden Dollar pro Jahr.* Online Artikel. Juni 2002. – www.heise.de/newsticker/meldung/28604

[Hochschulrechenzentrum 2003] Hochschulrechenzentrum, Uni Wuppertal: *MS-Windows und Verwandte.* Online Artikel. Oktober 2003. – www.hrz.uni-wuppertal.de/dienste/software/os/windows/

[Katz und Shapiro 1985] Katz, Michael L. ; Shapiro, Carl: Network Externalities, Competition, and Compatibility. In: *American Economic Review* 75 (1985), Nr. 3, S. 424–440

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

[Kotkamp 2001] Kotkamp, Stefan: *Electronic Publishing. Ökonomische grundlagen des Handels mit Informationsprodukten.* Karlsruhe, Universität Fridericiana zu Karlsruhe, Dissertation, 2001

[Küchlin und Weber 2005] Küchlin, Wolfgang ; Weber, Andreas: *Einführung in die Informatik.* Berlin [u.a.] : Springer, 2005

[Loasby 1999] Loasby, Brian J.: *Knowledge, Institutions, and Evolution in Economics.* London : Routledge, 1999

[Lopatka und Page 1999] Lopatka, John E. ; Page, Wiliam H.: Network Externalities. In: Bouckaert, Boudwijn (Hrsg.) ; De Geest, Gerrit (Hrsg.): *Encyclopedia of Law and Economics.* Cheltenham [u.a.] : Edward Elgar Publishing Limited, 1999, S. 952–980. – encyclo.findlaw.com/lit/0760book.pdf

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

[NIST und Technology 2002] NIST, The National Institute of S. (Hrsg.) ; Technology (Hrsg.): *The Economic Impacts of Inadequate Infrastructure For Software Testing. Final Report.* 2002. (Planning Report). – www.nist.gov/director/prog-ofc/report02-3.pdf

[Open Source Initiative 2004] Open Source Initiative: *The Open Source Definition.* Online Dokumrnt. 2004. – www.opensource.org/docs/definition_plain.php

[operating-system.org 2004] operating-system.org: *Windows Family.* Online Artikel. Juli 2004. – www.operating-system.org/betriebssystem/_german/bs-windows.htm

[Pasche und v. Engelhardt 2004] Pasche, Markus ; Engelhardt, Sebastian v.: Volkswirtschaftliche Aspekte der Open-Source-Softwareentwicklung. / Friedrich Schiller Universität. Jena, 2004 (18). – Jenaer Schriften Zur Wirtschaftswissenschaft

[Pasche und v. Engelhardt 2006] Pasche, Markus ; Engelhardt, Sebastian v.: Führt Open-Source-Software zu ineffizienten Märkten? In: Lutterbeck, B. (Hrsg.) ; Bärwolff, M. (Hrsg.) ; Gehring, R.A. (Hrsg.): *Open-Source-Jahrbuch 2006.* 2006, S. 93–108

[Persson und Siering 1999] Persson, Christian ; Siering, Peter: Big Brother Bill. Microsofts heimliche ID-Nummern - angeblich eine Panne. In: *c't - Magazin für Computertechnik* 6 (1999), S. 16–20

[Quah 2003] Quah, Danny: Digital Goods and the New Economy. / London School of Economics. London, 2003 (563). – CEP Discussion Papers. cep.lse.ac.uk/pubs/download/dp0563.pdf

[Raymond 1999] Raymond, Eric S.: *The Magic Cauldron.* Paper. 1999. – www.catb.org/ esr/writings/magic-cauldron/magic-cauldron.ps

[Romberg 2003] Romberg, Tim: Herstellerübergreifende Wiederverwendung von Komponenten. In: *Handbuch zur komponentenbasierten Softwareentwicklung.* -, 2003. – app2web.fzi.de/themen/ap4/cbse_handbuch.pdf

[Shapiro und Varian 1999] Shapiro, Carl ; Varian, Hal R.: *Information Rules. A Strategic Guide to the Network Economy.* Boston : Harvard Business School Press, 1999

[Shy 2001] Shy, Oz: *The Economics of Network Industries.* Cambridge : Cambridge University Press, 2001

[Spiegel Online 2000] Spiegel Online: *Katholiken misstrauen Scientology-Software.* Online Artikel. Juni 2000. – www.spiegel.de/netzwelt/politik/0,1518,79730,00.html

[Stelzer 2000] Stelzer, Dirk: Digitale Güter und ihre Bedeutung in der Internet-Ökonomie. In: *WiSt - Wirtschaftswissenschaftliches Studium* (2000), Nr. 6, S. 835–842

[tecChannel News 2003] tecChannel News: *Apache-Code ist kommerziellen Produkten ebenbürtig.* Online Artikel. Juli 2003. – www.tecchannel.de/news/20030702/thema20030702-11120.html

[Windisch 1987] Windisch, Rupert: Privatisierung natürlicher Monopole. Theoretische Grundlagen und Kriterien. In: Windisch, Rupert (Hrsg.): *Privatisierung natürlicher Monopole im Bereich von Bahn, Post und Telekommunikation.* Tübingen : Mohr Siebeck, 1987, S. 1–146

Jenaer Schriften zur Wirtschaftswissenschaft

2006

1 Roland **Helm** und Michael **Steiner**: Nutzung von Eigenschaftsarten im Rahmen der Präferenzanalyse - Eine Meta-Studie, Diskussion und Empfehlungen.

2 Uwe **Cantner** and Jens J. **Krüger**: Micro-Heterogeneity and Aggregate Productivity Development in the German Manufacturing Sector.

3 Roland **Helm**: Implication from Cue Utilization Theory and Signalling Theory for Firm Reputation and the Marketing of New Products.

4 Simon **Renaud**: Betriebsräte und Strukturwandel.

5 Wolfgang **Schultze**: Anreizkompatible Entlohnung mithilfe von Bonusbanken auf Basis des Residualen Ökonomischen Gewinns.

6 Susanne **Büchner**, Andreas **Freytag**, Luis G. **González**, Werner **Güth**: Bribery and Public Procurement - An Experimental Study.

7 Reinhard **Haupt**, Martin **Kloyer** und Marcus **Lange**: Patent indicators of the evolution of technology life cycles.

8 Wolfgang **Domschke** und Armin **Scholl**: Heuristische Verfahren.

9 Wolfgang **Schultze** und Ruth-Caroline **Zimmermann**: Unternehmensbewertung und Halbeinkünfteverfahren: Der Werteinfluss des steuerlichen Eigenkapitals.

10 Jens J. **Krüger**: The Sources of Aggregate Productivity Growth - U.S. Manufacturing Industries, 1958-1996.

11 Andreas **Freytag** and Christoph **Vietze**: International Tourism, Development and Biodiversity: First Evidence.

12 Nils **Boysen**, Malte **Fliedner**, Armin **Scholl**: A classification of assembly line balancing problems.

13 Wolfgang **Kürsten**: Offenlegung von Managergehältern und Corporate Governance - Finanzierungstheoretische Anmerkungen zur aktuellen Kapitalismusdebatte.

14 Sebastian v. **Engelhardt**: Die ökonomischen Eigenschaften von Software.

2005

1 Reinhard **Haupt**: Patent analysis of a company's technology strength.

2 Axel **Braßler**, Christoph **Grau**, Herfried **Schneider**: Wissenslabor Betriebswirtschaft - Eine Lehr-, Lern- und Kommunikationsumgebung für die universitäre und betriebliche Aus- und Weiterbildung.

3 Wolfgang **Kürsten**: Risikomanagement und aktionärsorientierte Unternehmenssteuerung - Mehr Fragen als Antworten.

4 Roland **Helm**, Reinhard **Meckl**, Nicole **Sodeik**: Wissensmanagement – Ein Überblick zum Stand der empirischen Forschung.

5 Uwe **Cantner**, Kristina **Dreßler**, Jens J. **Krüger**: Knowledge and Creative Destruction over the Industry Life Cycle - The Case of the German Automobile Industry.

6 Reinhard **Meckl**, Robert **Schramm**: Empirical evidence for a theory of international new ventures.

7 Andreas **Freytag**, Dirk **Schiereck**, Thomas W. **Thomas**: Consolidation and Market Power of Energy Utilities - The case of US-American and German Utility Takeovers.

8 Roland **Helm** und Oliver **Mauroner**: New Firms from Research-based Spin-offs.

9 Werner **Jammernegg** und Peter **Kischka**: A Decision Rule Based on the Conditional Value at Risk.

10 Roland **Helm** und Wolfgang **Stölzle**: Out-of-Stocks im Handel: Einflussfaktoren und Kundenreaktionsmuster.

11 Uwe **Cantner**, Kristina **Dreßler**, Jens J. **Krüger**: Knowledge Compensation in the German Automobile Industry.

12 Volkmar **Botta**, Martin **Köhler**: Zur Wertaufhellungskonzeption nach IAS 10.

13 Roland **Helm** und Michael **Gehrer**: Zum Aufbau von Vertrauen in interaktiven Entscheidungsprozessen.

14 Andreas **Feytag** and Donato **Masciandaro**: Financial Supervision Fragmentation and Central Bank Independence: The Two Sides of the Same Coin?

15 Volkmar **Botta** und Adrian A. **Weinaug**: Behandlung von Investitionszulagen für Sachanlagen gemäß Investitionszulagengesetz bei freiwilliger Offenlegung des Einzelabschlusses nach § 325 Abs. 2a HGB.

16 Volkmar **Botta** und Martin **Köhler**: Implikationen der Bestimmung des Cashflows aus betrieblicher Tätigkeit nach der direkten Methode.

17 Uwe **Cantner**, Andreas **Nicklisch** und Torsten **Weiland**: Innovation races: An experimental study on strategic research activities.

18 Markus **Pasche**: (Self-)Regulation of a Natural Monopoly via Complementary Goods -the Case of F/OSS Business Models.

19 Markus **Pasche**: Das Vertrauensspiel – eine verhaltensorientierte Erklärung.

20 Reinhard **Haupt** und Matthias **Korgel**: A Comparison between US and International Patent Classification as Input Data of a Technology Competition-Oriented Cluster Analysis.

21 Wolfgang **Kürsten**, Reinhard **Meckl** und Andreas **Krostewitz**: Value-Based M&A-Management – der M&A-Prozess im Lichte des Shareholder Value-Prinzips.

22 Simone **Martin**: Risikominimierung bei der Arbeitgeberwahl.

23 Wolfgang **Kürsten**: Neoklassische Finanzierungstheorie - Eine didaktisch motivierte Einführung.

24 Karsten **Korsa** und Simone **Martin**: Die Glaubwürdigkeit personalpolitischer Maßnahmen zur Signalisierung von Arbeitgeberattraktivität.