Universität Tübingen
Lehrstuhl für Bürgerliches Recht,
Handels- und Wirtschaftsrecht
Prof. Dr. iur. Wernhard Möschel

Open Source – ein Widerspruch zu den Grundlagen des Urheberrechts?

Seminararbeit im Rahmen des Seminares

„Internet- und Informationsrecht“

im Sommersemester 2002

von Michael Kubert

Literatur

Balzert, Helmut: Lehrbuch der Software-Technik, Heidelberg 1996

Bechtold, Stephan: Vom Urheber- zum Informationsrecht, München 2002

Beckmann, Heiner: EDV-Anwenderdokumentationen, CR 1998, S. 519 ff.

Bergmann, Margarethe / Pötter, Godehard / Streitz, Siegfried: Handbücher für Softwareanwender, CR 2000, 555 ff.

Brandt, Jochen C.: Bewertungskriterien für Anwenderhandbücher, CR 1998, S. 571 ff.

Brooks, Frederick P. jr.: The Mythical Man-Month, Boston 1995

Bundesministerium für Bildung und Forschung: Analyse und Evaluation der Softwareentwicklung in Deutschland, Dezember 2000, http://www.dlr.de/IT/IV/Studien/evasoft_abschlussbericht.pdf, [BMBF]

Bundesministerium für Wirtschaft und Technologie: Mikro- und makroökonomische Implikationen der Patentierbarkeit von Softwareinnovationen, September 2001, http://www.bmwi.de/Homepage/download/technologie/Softwarepatentstudie.pdf [BMWi]

Carnegie Mellon University, Software Engineering Inst.: Capability Maturity Model, Boston 1995

Christensen, Clayton M.: The Innovator's Dillemma, New York 2000

De Marco, Tom: Why Does Software Cost So Much?, New York 1995

DiBona, Chris / Ockman, Sam / Stone, Mark (Hrsg.): Open Sources, Sebastopol 1999

Dröschel, Wolfgang / Wiemers, Manuela (Hrsg.): Das V-Modell 97, München 2000

Erman, Walter: Bürgerliches Gesetzbuch, 10. Auflage, Köln 2000

Fechner, Frank: Geistiges Eigentum und Verfassung, Tübingen 1999

Ghosh, Rishab / Prakash, Vipul Ved: The Orbiten Free Software Survey, 2000, http://firstmonday.org/issues/issue5_7/ghosh/index.html

Grützmacher, Malte: Open-Source-Software, die GNU General Public License, ITRB 2002, S. 89 ff.

Günter, Andreas: Auktionen im Internet, ITRB 2002, S. 93 ff.

Haft, Frietjof: Verhandeln, die Alternative zum Rechtsstreit, München 1992

Hoch, Detlef J. / Roeding, Cyriac R. / Purkert, Gert / Lindner, Sandro K.: Erfolgreiche Software-Unternehmen, München 2000

Jacobson, Ivar / Booch, Grady / Rumbaugh, James: The Unified Software Development Process, Boston 1998

Jaeger, Till / Metzger, Axel: Open Source Software, München 2002

Koch, Frank A.: Urheber- und kartellrechtliche Aspekte der Nutzung von Open-Source-Software (I), CR 2000, S. 273 ff.

Koch, Frank A.: Urheber- und kartellrechtliche Aspekte der Nutzung von Open-Source-Software (II), CR 2000, S. 333 ff.

Lerner, Josh / Tirole, Jean: The Simple Economics of Open Source, 2000, http://www.people.hbs.edu/jlerner/simple.pdf

Lessig, Lawrence: The Future Of Ideas, New York 2001

Litman, Jessica: Digital Copyright, New York 2001

Louden, Kenneth C.: Programmiersprachen, Bonn 1994

Marly, Jochen: Softwareüberlassungsverträge, 3. Aufl., München 2000

McGowan, David: Legal Implications of Open-Source Software, Illinois 2001

Moglen, Eben: Anarchism Triumphant, Free Software and the Death of Copyright, 1999, http://firstmonday.org/issues/issue4_8/moglen/index.html

Möhring, Philipp / Nicolini, Käte (Hg.): Urheberrechtsgesetz, München 2000

Moore, Geoffrey A.: Crossing the Chasm, New York 1999

Raymond, Eric S.: The Cathedral & The Bazaar, Sebastopol 2001

Rechenberg, Peter / Pomberger, Gustav: Informatik-Handbuch, 2. Aufl., München 1999

Rössel, Markus: Patentierung von Computerprogrammen, ITRB 2002, S. 90 ff.

Sandl, Ulrich: „Open-Source“-Software: Politische, ökonomische und rechtliche Aspekte, CR 2001, S. 346 ff.

Schäfer, Hans-Bernd / Ott, Claus: Lehrbuch der ökon. Analyse des Zivilrechts, 3. Aufl., Berlin 2000

Scheer, August-Wilhelm: Wirtschaftsinformatik, 2. Auflage, Berlin 1998

Schneider, Uwe / Werner, Dieter: Taschenbuch der Informatik, 4. Auflage, Leipzig 2001

Schricker, Gerhard (Hrsg.): Urheberrecht, 2. Aufl., München 1999

Sester, Peter: Open-Source-Software: Vertragsrecht, Haftungsrisiken und IPR-Fragen, CR 2000, S. 797 ff.

Thaller, Georg Erwin: Software-Qualität, Berlin 2000

Torvalds, Linus / Diamond, David: Just for Fun, New York 2001

Wadle, Elmar: Geistiges Eigentum, Weinheim 1996

Weinberg, Gerald M.: The Psychology of Computer Programming, New York 1998

Williams, Sam: Free as in Freedom, Sebastopol 2002

Zahrnt, Christoph: Projektmanagement von IT-Verträgen, Heidelberg 2001

Zahrnt, Christoph / Erben, Meinhard: IT-/DV-Verträge, 2. Auflage, Heidelberg 2001

Open Source scheint die Grundannahmen des Urheberrechts auf den Kopf zu stellen. Es wird untersucht, welche Annahmen dem Urheberrecht zu Grunde liegen. Sodann wird aufgezeigt, welche Anreize und Hemmnisse den Markt für Software im Allgemeinen und für Open-Source im Besonderen bestimmen. Es stellt sich die Frage, inwiefern Open-Source ein Widerspruch zu den Grundannahmen des Urheberrechts darstellt. Es wird diskutiert, ob Open-Source zu einem tauglichen Modell einer effizienten, dezentralisierten Informationsproduktion ausgebaut werden kann. Anschliessend werden Alternativen zu Open-Source besprochen, die eine Lösung der Open-Source fördernden Probleme bringen könnten. Das Fazit fasst die Ergebnisse zusammen.

A. Grundannahmen des Urheberrechts

I. Historische Wurzeln

1. Zuordnung des Werkes zu seinem Urheber

Der Gedanke, dass ein Werk seinem Urheber zugeordnet sein soll, ist ein sehr abstrakter Gedanke, dessen Ursprung sich nicht terminieren lässt[1]. „Geistiges Eigentum“ im juristischen Sinne basiert auf dieser Zuordnung[2]. Die Zuordnung war gesellschaftlich nicht immer und für alle Werktypen existent[3]. Noch bis in die Renaissance[4] hinein wurden Handwerker nicht als Künstler, sondern Werkzeuge angesehen, die nach Verrichtung ihrer Arbeit in die Anonymität zurückkehrten; der Beitrag der Einzelnen am Gesamtprojekten erschien jeweils unbedeutend[5]. Im Laufe der Zeit trat das mutmassliche Urbedürfnis in den Vordergrund, die eigene Person durch das Werk zu verewigen[6]. Der Schöpfer könnte allerdings auch früher in einer kleinen Gruppe bereits ausreichend bekannt gewesen sein, so dass es keiner expliziten Namensnennung bedurfte[7].

1. Vgl. Fechner S. 18.
2. Vgl. Fechner S. 18.
3. Z.B. nicht bei Werken, die als Gebrauchsgegenstände angesehen wurden, zur allgemeinen Verbesserung der Lebensverhältnisse geschaffen wurden oder mit denen keine Wirkung auf andere Menschen beabsichtigt war, vgl. Fechner S. 19.
4. Etwa vom Jahr 1430 bis 1600, umfasst die Wende vom Mittelalter zur Neuzeit, an die Stelle des Autoritätsglaubens tritt der Geist kritischer Forschung; der Mensch wird zum Maß aller Dinge.
5. Vgl. Fechner S. 19. Hier gibt es Parallelen zur „Open-Source-Gemeinde“.
6. Vgl. Fechner S. 19 f., er erwähnt Eitelkeit, Stolz, Streben nach Unsterblichkeit und Wettbewerbsvorteile als Motive. Vor dieser Zeit gab es nur selten erhaltene Namensnennungen, z.B. bei Homer. Gerade die Verewigung kommt aber bei den schnellen Alterungszyklen von Software wohl nicht in Betracht.
7. Vgl. Fechner S. 21.

2. „Geistiger Diebstahl“

Schon in der griechischen Antike warfen sich Dichter und Philosophen gegenseitig „geistigen Diebstahl“ vor, und es gab Fälle, in denen ein Dichter fremde Werke als die eigenen ausgab und dadurch anderen ausgelobte Preise wegnahm[1].

3. Wettstreit

Auch sind aus der Antike bereits Werkinschriften bekannt, die darauf hinweisen, dass der Künstler einen anderen Künstler reizen wollte, mit ihm in einen Wettstreit zu treten[2].

4. Arbeitsteilung

Seit Beginn der Arbeitsteilung müssen die Urheber von anderen Mitgliedern der Gesellschaft versorgt worden sein[3]. Das spricht für eine besondere Stellung der Künstler in der Gesellschaft.

5. Manuskriptverkauf

Zur Zeit des römischen Rechts war bereits der Manuskriptverkauf bekannt; Honorarzahlungen gab es allerdings nur bei der Erstveröffentlichung, die weitere Verbreitung war frei[4]. Es gab auch hier noch kein normiertes Schutzsystem[5]. Zwar wurde der physische Gegenwert des Werkes entlohnt, im übrigen war der Urheber jedoch auf freiwillige Belohnung verwiesen[6].

6. Frühe Kopierfirmen

Bereits in der Antike waren Kopierfirmen bekannt, die durch gleichzeitiges Diktieren von Texten an mehrere Schreiber bis zu 1000 Exemplare produzieren konnten[7]. Dennoch gab es noch kein Schutzsystem für die wirtschaftlichen Interessen der Urheber; vermutlich bestand dafür keine wirtschaftliche Notwendigkeit[8].

7. Recht des Urhebers über seine Werke

Thomas von Aquin[9] war einer der Bedeutendsten, der ein Eigentums-, ein Herrschafts- und Verfügungsrecht des Menschen über die von ihm geschaffenen Werke forderte[10]. Rechtlich war dies jedoch noch nicht normiert, es gab lediglich die „Bücherflüche“[11].

1. Vgl. Fechner S. 21.
2. Vgl. Fechner S. 22.
3. Vgl. Fechner S. 21 m.w.N.
4. Vgl. Fechner S. 23; auch hier drängt sich der Vergleich zu Open-Source auf.
5. Vgl. Fechner S. 23.
6. Vgl. Fechner S. 23.
7. Vgl. Fechner S. 23 m.w.N.
8. Die Persönlichkeitsrechte wurden dagegen geachtet, vgl. Fechner S. 23 f. m.w.N.
9. Von 1225-1274.
10. Vgl. Fechner S. 25 m.w.N.
11. Eine Art Vorwort, in dem man (Falsch-)Kopierer verfluchte, vgl. Fechner S. 25.

8. Privilegienwesen

Im Bereich der Erfindungen argumentierte man damit, dass der Schöpfer für die Öffentlichkeit nützliche Dinge gefunden habe, aber in dieser Zeit keine anderweitigen Gewinne erzielen konnte[1]. Daher bedürfe es eines Ausgleichs. Dies waren frühe Forderungen nach einem Privilegienwesen, wie es seit Beginn des 16. Jahrhundert in Europa entstand[2], in China scheint es schon um das Jahr 1185 entstanden zu sein[3]. Grundsätzlich waren alle Werke frei nachdruckbar, bis eben auf die weniger privilegierter Autoren[4]. Geschützt wurden primär die Verleger bzw. Drucker; Autorenhonorare wurden erst Mitte des 16. Jahrhunderts allgemein üblich[5]. Der Drucker sollte aus dem Kapital, mit dem er den Druck vorfinanzierte, einen Gewinn erwirtschaften können[6]. Bei Privilegienerteilung an den Autor lag der Grund in seiner handwerklichen Arbeit, seinem Fleiss und der von ihm aufgewendeten Zeit, nicht jedoch der geistigen Leistung[7].

9. Geistiges Eigentum

Erst um 1738 kommt massgeblich Johann Rudolf Thurneysen mit Hilfe einer konsequenten Anwendung des Naturrechts zur Überzeugung, dass bereits unabhängig von wirtschaftlichen Aspekten dem Urheber ein Eigentumsrecht an seinem Werk zustehe, welches ihm auch ermöglichen solle, andere von der Nutzung auszuschliessen[8]. Er verweist ferner darauf, dass Gelehrte durch unerlaubten Nachdruck weniger Lohn erhielten und daher nicht mehr bereit wären, Bücher zu schreiben[9]. Allerdings sah er den Nachdruck überteuerter Bücher als rechtmässig an, nämlich als Mittel gegen den Missbrauch der Druckkunst[10]. Kant schliesslich forderte 1785, dass das Nachdruckrecht nicht aus dem dinglichen Besitz eines Werkexemplares gefolgert werden könne, sondern separat vom Autor erworben werden müsse[11]. Er hält jedoch Bearbeitungen und Übersetzungen auch ohne Zustimmung für legitim[12]. Fichte

1. Vgl. Fechner S. 26 m.w.N. Die Verfechter freier Software scheinen heute vergessen zu haben, dass andere in der Zeit der Werkerstellung Geld verdienen können.
2. Vgl. Fechner S. 26 m.w.N.
3. Vgl. Fechner S. 27 m.w.N.
4. Vgl. Fechner S. 27 m.w.N.
5. Vgl. Fechner S. 26 bzw. 27 m.w.N.
6. Vgl. Fechner S. 30 m.w.N.
7. Vgl. Fechner S. 30 f. m.w.N.
8. Vgl. Fechner S. 33 m.w.N.
9. Vgl. Fechner S. 34 m.w.N.
10. Vgl. Fechner S. 34 m.w.N.; überraschenderweise wird dieses Argument heute auch in Kreisen sog. „Raubkopierer“ genannt.
11. Vgl. Fechner S. 35 m.w.N.
12. Vgl. Fechner S. 36 m.w.N.

schliesslich unterschied 1793 bereits Inhalt und Form des Werkes. So werde beispielsweise das wörtliche Abdrucken von Vorlesungen nicht geduldet (Form), wohl aber das Zueigenmachen und Verbreiten ihrer Gedanken (Inhalt)[1]. Er wendet ein, dass man zwar auf den Erwerb der Verwertungsrechte verzichten könne, sie sich aber nicht einfach aneignen dürfe, wenn sie einem zu teuer seien[2]. Hegel erwähnt 1833 in einem posthum veröffentlichten Werk, dass Wissenschaft und Kunst gefördert werden, wenn der geistige Diebstahl unterbunden werde[3]. Die Künstler werden zu dieser Zeit von der Bevormundung durch Mäzene entbunden, zugleich aber in wirtschaftliche Abhängigkeit vom Kunstmarkt geführt; die Zahlung einer Belohnung durch den Mäzen wird durch eine marktwirtschaftliche Vergütung ersetzt[4]. Die Theorie des geistigen Eigentums koppelt das Nutzungsrecht nicht mehr an die Refinanzierung des Verlegers, sondern an das Bestimmungsrecht des Urhebers[5]. John Locke legte mit seiner Arbeitstheorie den Grundstein für die Lehre vom geistigen Eigentum[6]. Danach darf ein Mensch das Ergebnis seiner Arbeit als sein Eigentum beanspruchen. In einigen Ländern wurde ein positivrechtlicher Schutz mit volkswirtschaftlichen Gründen, etwa zur Förderung der allgemeinen Volksbildung, propagiert[7]. Im Jahre 1709 bezeichnete das „Statute of Anne“ als erstes modernes Urheberrechtsgesetz die Förderung des Lernens als Ziel, gelehrte Männer sollten ermutigt werden, nützliche Bücher zu schreiben[8]. Der Schutz des Individuums wird so durch das Allgemeininteresse gerechtfertigt[9]. Anlässlich einer unerlaubten Nachschrift einer fehlerhaft wiedergegebenen Vorlesung wurde in den Niederlanden der Nachdruck ohne Erlaubnis des Verfassers verboten[10]. Goethe klagte über finanzielle Einbussen durch Nachdrucke und fühlte sich dadurch „vogelfrei“ und seine Rechte dem „Handwerker und Fabrikanten unbedingt preisgegeben“[11].

10. Privatkopie

Im Jahr 1901 wird ein Reichsgesetz erlassen, welches die Vervielfältigung ohne Einwilligung des Berechtigten unabhängig vom Kopierver-

1. Vgl. Fechner S. 37 m.w.N.
2. Vgl. Fechner S. 37 bis 38 m.w.N.
3. Vgl. Fechner S. 38 m.w.N.
4. Vgl. Fechner S. 39 m.w.N.
5. Vgl. Fechner S. 40 m.w.N.
6. Vgl. Fechner S. 41 m.w.N.
7. Vgl. Fechner S. 46 m.w.N.
8. Vgl. Fechner S. 46 m.w.N.
9. Vgl. Fechner S. 46.
10. Vgl. Fechner S. 47 m.w.N.; vgl. die heutige Handhabung bei Open-Source.
11. Zitiert nach Fechner S. 47 m.w.N.

fahren verbietet, aber auch eine Vervielfältigung zum persönlichen Gebrauch gestattete, solange mit ihr keine Einnahmen aus dem Werk erzielt werden sollten[1]. Man ging davon aus, dass dem Urheber durch die tatsächlichen Verhältnisse kein Schaden durch die Regelung entstehen würde[2].

11. Wirtschaftliche Beteiligung des Urhebers

In der Weimarer Republik kam das Reichsgericht in seiner Rechtsprechung zu einer urheberfreundlichen Auslegungsregel, welche besagte, dass der Urheber an allen Arten der wirtschaftlichen Nutzung des Werkes zu beteiligen sei[3]. Darüber hinaus wurde im Wettbewerbsrecht ein allgemeines Nachahmungsverbot verankert, weil dadurch dem Nachahmer die Gewährung günstigerer Preise und somit die wirtschaftliche Schädigung des Erzeugers möglich sei[4]. Später rückte man von dieser Regelung im Interesse des gesellschaftlichen Fortschritts ab[5].

12. „Gemeinnutz geht vor Eigennutz“

In der Zeit des Nationalsozialismus wurde daraus die Zulässigkeit von gesetzlichen und Zwangslizenzen gefolgert[6]. Man betonte ideologisch, dass die Urheberinteressen der Allgemeinheit, insbesondere der Anhebung des Bildungsniveaus und der allgemeinen Kultur, schadeten[7]. Das Werk sei ein Gut der Volksgemeinschaft, da der Urheber aus dem Allgemeingut schöpfe[8].

13. Landesverfassungen und Grundgesetz

Die Landesverfassungen vor Erlass des Grundgesetzes betonen hingegen wieder den Arbeitscharakter der urheberlichen Schöpfung[9]. Der Parlamentarische Rat schien von der eigentumsrechtlichen Garantie des geistigen Eigentums als selbstverständlich auszugehen, wohl deswegen fand es keine explizite Erwähnung im Grundgesetz[10]. Der BGH greift zunächst auf naturrechtliche Begründungen zurück[11].

14. Entwicklung unter dem Grundgesetz

In der Begründung des Urheberrechtsentwurfs von 1962 wurde betont, dass der Urheber gegen unbefugte wirtschaftliche Verwertung seiner

1. Vgl. Fechner S. 55 m.w.N.
2. Vgl. Fechner S. 55 m.w.N.; das Kopieren war sehr aufwändig.
3. Vgl. Fechner S. 56 m.w.N.
4. Vgl. Fechner S. 56 m.w.N.
5. Vgl. Fechner S. 56 m.w.N.
6. Vgl. Fechner S. 58 m.w.N.
7. Vgl. Fechner S. 57 m.w.N.
8. Vgl. Fechner S. 58 m.w.N.
9. Vgl. Fechner S. 58 m.w.N.
10. Vgl. Fechner S. 60 m.w.N.
11. Vgl. Fechner S. 63 m.w.N.

schöpferischen Leistung zu schützen sei[1]. Im Gesetz von 1965 wurde das Urheberrecht in Reaktion auf neue Kopiermöglichkeiten als ein umfassendes absolutes Recht gestaltet, welches dem Urheber auch künftige Verwertungsmöglichkeiten vorbehält und seine ideellen Interessen schützt[2]. Es erfolgt gemäss der monistischen Theorie die Trennung von Urheberrecht (nicht übertragbar) und Verwertungsrechten (übertragbar)[3]. Die Vervielfältigung zum persönlichen Gebrauch ist nicht untersagbar, zur Abgeltung erhält der Urheber einen Vergütungsanspruch[4]. Die Reform des Urheberrechts von 1983 stellte unter dem Einfluss der Fotokopiertechnik fest, dass die Bedeutung der vergütungsfreien Kopierhandlungen seit 1965 deutlich angewachsen war, so dass man diese als neue Nutzungsart ansehen müsse[5]. Der Urheber solle überall dort beteiligt werden, wo aus seinem Werk wirtschaftlicher Nutzen gezogen wird[6]. Die Entstehung weiterer Schutzgesetze war die Folge[7].

15. Rechtsschutz von Computerprogrammen

Zunächst verlangte der BGH für die Schutzfähigkeit eines Programmes eine besondere Gestaltungshöhe[8]. Dadurch unterlagen nur verhältnismässig wenige Programme dem Urheberrechtsschutz[9]. Der Einfluss des Europarechts führte zur Fortentwicklung des Urheberrechts im Jahre 1995[10]. Seitdem geniessen Computerprogramme den vollen Schutz des Urheberrechts, unabhängig von der Gestaltungshöhe[11]. Man spricht auch vom „Leistungsschutz im urheberrechtlichen Gewand“[12]. Ferner gibt es kein Recht auf eine Privatkopie für Programme[13].

16. Zusammenfassung

Die historische Entwicklung verlief vom Drucker- und Verlegerschutz hin zum Schutz des Urhebers. Zentraler Punkt dabei ist die Refinanzierung der Werkverbreitung bzw. Werkschöpfung. Während anfangs wettbewerbliche Überlegungen überwogen, steht heute die Belohnung

1. Vgl. Fechner S. 63 m.w.N.
2. Vgl. Fechner S. 63 m.w.N.
3. Vgl. Fechner S. 64 m.w.N.
4. Vgl. Fechner S. 64 m.w.N.
5. Vgl. Fechner S. 64 m.w.N.
6. Vgl. Fechner S. 64 m.w.N.
7. Z.B. Marken-, Geschmacksmuster-, Gebrauchsmuster-, Sortenschutz-, Halbleiterschutzgesetz, vgl. Fechner S. 65 m.w.N.
8. Vgl. Fechner S. 66 m.w.N.
9. Vgl. Fechner S. 66 m.w.N.
10. Vgl. Fechner S. 66 m.w.N.
11. Vgl. zur vorherigen Rechtslage, insbesondere zur Anforderung, das Programm müsse „das Können eines Durchschnittsprogrammierers deutlich überragen“ auch Möhring/Nicolini/Hoeren vor § 69 a Rn 1.
12. Vgl. Möhring/Nicolini/Hoeren § 69 a Rn 16.
13. Da § 69 c UrhG lex specialis zu § 53 UrhG ist, vgl. Möhring/Nicolini/Decker § 53 Rn 1 und Schricker/Loewenheim § 53 Rn 10.

des Urhebers für die Werkschöpfung im Mittelpunkt. Die Entwicklung wurde begleitet von der Überlegung, dass Missbrauch verhindert und das Allgemeinwohl gefördert werden sollte.

II. Schaffung von Anreizen durch Property Rights

1. Erhöhte Refinanzierungschance

Man geht davon aus, dass ein Urheber die Schaffung des Werkes unterlässt, wenn er nicht eine überzeugende Chance[1] hat, wenigstens die Kosten[2] der Werkschöpfung durch eine folgende Vermarktung decken zu können (und möglichst auch Aussicht auf Gewinn hat[3]). Der Urheber, der aus Altruismus handelt und seine Werke verschenkt, wird bei dieser Betrachtung ignoriert[4]. Historisch war weniger der Aufwand für das Verfassen der Texte zu schützen, denn der folgende Distributionsprozess durch die Druckereien und Verlage. Wenn man von vermögenden Autoren absieht, boten die Verlage den Urhebern überhaupt erst die Möglichkeit, ihre Werke zu verfassen, da sie zumindest die Lebenshaltungskosten für die Zeit der Werkerstellung finanzierten[5]. Hinzu kommt die Einsicht, dass altruistische Urheber selten sind, jedenfalls wenn man sie in Relation zum Bedarf an Werken setzt.

2. Durch Vergabe von Property Rights

Der Gesetzgeber hat daher den Urhebern ein Monopol eingeräumt. Sie können im wesentlichen exklusiv über die Verbreitung ihres Werkes bestimmen. Diese Verwertungsrechte können sie z.B. an Verlage abtreten. Das Monopol wird gewährt, weil der Urheber mit seinem Werk die Öffentlichkeit kulturell bereichert, und damit der Gesellschaft etwas zurück gibt, wobei man mangels Maßstab auf eine Bewertung verzichtet. Ansonsten wäre das Monopol systemwidrig, denn die meisten Ideen und Anregungen, die der Urheber für sein Werk nutzt, stammen aus der Public Domain, also dem Fundus gemeinfreien Wissens. Die Bewertung der Nützlichkeit des Werkes wird dem Markt überlassen[6].

1. Niemand mag garantieren, dass die Refinanzierung gelingt. Der Faktor Nützlichkeit spielt eine entscheidende Rolle. Es soll schliesslich nicht Arbeit, sondern (für andere nützliche) Leistung belohnt werden; vgl. Fechner S. 106.
2. Historisch bedeutsam waren die Druckkosten (Druckerpressen, Satzvorlagen).
3. Man kann die Opportunitätskosten betrachten. Statt das Werk zu schaffen, könnte der Urheber sein Geld auch anlegen. Aus wirtschaftlicher Sicht muss also mindestens ein Gewinn in Höhe der Zinsen wahrscheinlich sein.
4. Weil man davon ausgeht, dass er keines wirtschaftlichen Schutzes bedürfe.
5. Vgl. das Mäzenatentum.
6. Das ist nicht ganz unkritisch, wie der Kunstmarkt zeigt, denn „objektiv“ hochwertige Werke werden ggf. nicht entsprechend honoriert (z.B. aufgrund von Modetrends). Man versucht dem beispielsweise durch staatliche Fördermassnahmen entgegen zu wirken, um das Potential der Werkschöpfer unter dem Einfluss der Überlegung zu erhalten, dass ihre Fähigkeiten später einmal vom Markt benötigt werden könnten.

3. Zusammenfassung

Eine wirtschaftliche Grundannahme des Urheberrechts ist, dass die Qualität, Quantität und Nützlichkeit von Werkschöpfungen durch Anreize gefördert werden kann. Zu diesem Zwecke werden dem Urheber Property Rights gewährt.

III. Zusammenfassung

Die historische Entwicklung hat zur Vergabe von Property Rights an die Werkschöpfer geführt, um ihnen bessere Refinanzierungschancen zu ermöglichen.

B. Der Software-Markt und Open-Source

I. Vorbemerkung

Die Verfügbarkeit von Open-Source-Software könnte darauf hindeuten, dass die Anreizwirkung der Property Rights im Urheberrechts zwar potentiell vorhanden ist, aber nicht benötigt wird. Daher erläutert dieser Abschnitt die Aspekte, die eine Entwicklung von Software im Allgemeinen und Open-Source-Software im Besonderen reizen oder hemmen. Im Vordergrund stehen dabei qualitative Zusammenhänge, nicht quantitative Aspekte. Dazu werden zahlreiche Fallbeispiele genannt, um die teilweise komplexen Wechselwirkungen zu verdeutlichen.

II. Begriffsbestimmung

1. Software

Der Begriff „Software“ ist recht unbestimmt[1]. Programme und Daten sind schwer auseinander zu halten[2]. Ein Ansatz zur Definition von Software verlangt, dass sie das Ergebnis einer eigenen geistigen Schöpfung ihres Urhebers sein muss[3]. Diese Definition erfasst einen grossen Teil aller von Software generierten Daten fälschlich auch selbst als Software[4]. Daher wird dieser Ansatz nicht weiter verfolgt.

Im folgenden soll unter „Software“ (und synonym dazu „Programm“)

1. § 69 a 1 UrhG verweigert sogar eine Definition, da diese zu schnell veralten könnte, vgl. Möhring/Nicolini/Hoeren § 69 a Rn 2.
2. Beispiel: Um ein Lied zu hören, kann man einen MP3-Player und eine MP3-Datei nehmen. Die MP3-Datei, die das Lied enthält, könnte man jedoch auch als Programm für den MP3-Player betrachten, den Player als eine Art von Betriebssystem für MP3-Dateien. Damit wäre die MP3-Datei als Software bezüglich der Ablaufumgebung MP3-Player zu verstehen. Das deckt sich aber nicht mit dem herkömmlichen Verständnis von Software und würde dazu führen, ggf. Probleme des Musikmarktes diskutieren zu müssen. Vgl. dazu die Definitionen von Schneider, Hesse und IEEE bei Balzert, S. 22. Diese umfassen neben Programmen explizit auch Daten.
3. Vgl. § 69 a II UrhG; Möhring/Nicolini/Hoeren § 69 a II Rn 2.

jede mit Hilfe einer üblichen Programmiersprache[1] erzeugte Folge von Anweisungen an einen Computer verstanden werden[2]. Eine Programmiersprache ist ein notationelles System zur Beschreibung von Berechnungen in durch Maschinen und Menschen lesbarer Form[3]. Auch dies ist ein weiter Begriff: Er umfasst Maschinensprachen, Assembler-Sprachen, höhere Programmiersprachen und anwendungsorientierte Sprachen[4]. Abweichend von der Definition sollen im Folgenden jedoch explizit Sprachen ausgenommen werden, die vorwiegend zur Generierung[5] und Beschreibung von Dokumenten benutzt werden[6]. Der Einfachheit halber wird folgend der Begriff „Software“ in dem soeben erläuterten Sinne verwendet[7].

Software lässt sich wiederum in verschiedene Gattungen einteilen[8]. Dazu kann die Hardwarenähe als Kriterium verwendet werden (Microcode, Betriebssystem, Textverarbeitungsprogramm), der Grad der Angepasstheit (Individualsoftware, angepasste Standardsoftware, zusammengesetzte Lösung aus Standard-Softwarekomponenten, vollständige Lösung durch Standardsoftware[9]), die Anwendungsbereiche (Produktion, Vertrieb, Verwaltung) oder die Branchenzugehörigkeit (Industrie, Handel, Handwerk, Dienstleistung, öffentlicher Sektor)[10].

2. Software-Erstellung

Wenn man den Begriff „Software-Erstellung“ eng auslegt, so ist damit lediglich die Erstellung des Quelltextes und folgende Übersetzung in ein maschinenlesbares Format gemeint, also die Implementierung. Im Rahmen dieser Arbeit wird der Begriff jedoch weiter verstanden. Er lässt sich gliedern in Anforderungsermittlung (Analyse), Systemdefini-

4. Weil sie auch ein (mittelbares) Ergebnis geistiger Schöpfung sind, z.B. eine normale Textdatei. Richtigerweise ausgeschlossen würden z.B. automatisch von Programmen generierte Protokolldateien, z.B. Messwerte. Als Ergebnis der geistigen Schöpfung ihres Urhebers kann man aber auch normale Textdateien auffassen. Es ist jedoch nicht sachgerecht, jeden Textschreiber als Softwareentwickler zu bezeichnen.
1. Das umfasst Sprachen aller Typen (prozedural, funktional, logisch, objektorientiert) wie Assembler, Cobol, C/C++, Java, Prolog, Lisp und Skriptsprachen.
2. Vgl. auch die Definition bei BMBF S. 30 f.
3. Louden S. 3.
4. Vgl. Schneider/Werner/Spielmann, 8.2.1.3, dort folgen zahlreiche Beispiele; Rechenberg/Pomberger/Goos/Zimmermann, D 2.1.3.
5. Vgl. Möhring/Nicolini/Hoeren § 69 a Rn 14.
6. Vgl. Schneider/Werner/Spielmann, 8.2.1.3, z.B. TeX, SGML, HTML, XML, PDF. Diese werden in der Regel von Programmen generiert. Nicht aber XSLT.
7. Ein Informatik-Laie hätte den Begriff vermutlich genau so verstanden und wäre gar nicht auf die Idee gekommen, eine Textdatei könne „Software“ sein.
8. Ein historischer Abriss findet sich bei Hoch/Roeding/Purkert/Lindner S. 281 ff.
9. Prozentual ergibt sich dafür etwa die Abstufung 5/10/35/50 %, mit zunehmendem Trend bei vollständig neuer Software hin zu Anpassungen, vgl. Balzert S. 33. Die Zeit der Massenmarktsoftware begann etwa im Jahr 1981 mit der Entwicklung des IBM PC, vgl. Hoch/Roeding/Purkert/Lindner S. 33.
10. Ein Überblick findet sich unter http://www.softguide.de.

tion, Entwurf, Implementierung (die eigentliche „Programmierung“), Validation (Testen), Systemeinführung (technische und Benutzer-Dokumentation) und Wartung[1]. Eine andere Gliederung geht von Analyse, Fachentwurf, DV-Entwurf, Implementierung, Integration, Konsolidierung aus[2]. All diese Tätigkeiten sind von der Informatik-Seite her zur Erstellung des Endproduktes Software notwendig.

3. Open-Source

Der Begriff[3] ist historisch als Verdeutlichung von „free software“ zu sehen[4]. Das Wort „free“ war missverständlich, da es sowohl im (gemeinten) Sinne von „Freiheit“ als auch (fälschlich) „kostenlos“ interpretiert werden konnte[5]. Im Kern geht es darum, dass der Nutzer Kopien erstellen und diese verbreiten darf, ein Recht auf den Quelltext des Programmes eingeräumt bekommt sowie diesen ändern und Verbesserungen am Programm vornehmen darf[6]. Die Änderungen dürfen ihrerseits z.T. auch unter einer anderen Lizenz verbreitet werden[7]. Die wesentlichen Lizenzoptionen sind GPL, LGPL, BSD, NPL, MPL und Public-Domain[8]. Bei einigen darf die Software nicht mit proprietärer Software vermischt werden. Sie unterscheiden sich auch in der Frage, ob Änderungen geheim gehalten werden dürfen oder selbst veröffentlicht werden müssen. Grundsätzlich erlaubt es Open-Source, dass der Urheber für seine Arbeit entlohnt wird. Nichtsdestotrotz wird diese Option des Open-Source-Modells in der Praxis äusserst selten genutzt. Sie stösst auch an Grenzen, da der Lizenznehmer seinerseits die Software kostenlos verbreiten könnte und damit eine relevante direkte Entlohnung des Urhebers unrealistisch ist[9].

4. Zusammenfassung

Software, Software-Erstellung und Open-Source wurden definiert.

III. Der Softwaremarkt im Allgemeinen

Die folgenden Feststellungen sind im Softwaremarkt von Bedeutung.

1. Vgl. Rechenberg/Pomberger/Floyd/Züllighoven, D 14.5.2.
2. Vgl. Schneider/Werner/Spielmann 9.2.1, Bild 9.2.
3. Vgl. zu seiner Entstehung am 03.02.1998: Raymond S. 175.
4. Vgl. Raymond S. 175 und Williams S. 164.
5. Von Stallman stammt die Erklärung „free as in free speech, not free beer“, vgl. auch DiBona/Ockman/Stone/Stallman S. 56 f.
6. Eine breitere Erklärung der Lizenzen findet sich bei DiBona/Ockman/Stone/Perens S. 171 ff.
7. Vgl. DiBona/Ockman/Stone/Perens S. 177.
8. Siehe den Vergleich bei DiBona/Ockman/Stone/Perens S. 185.
9. Darin gibt es historische Parallelen: „5. Manuskriptverkauf” auf Seite 2.

A 1. Software

1. lässt sich mit geringen Investitionen entwickeln

Die Markteintrittsbarrieren sind für Unternehmen, die Software entwikkeln wollen, vergleichsweise niedrig. So ist es mit unter $ 100.000 bereits möglich, ein erfolgreiches Software-Unternehmen zu gründen[1]. Das durchschnittliche Startkapital über alle Branchen liegt weltweit dagegen bei $ 5,1 Mio.[2]. Für eine Papierfabrik beispielsweise benötigt man rund $ 1 Milliarde, bevor die Produktion beginnt und schreibt danach jahrelang Verluste[3]. Zur Entwicklung von Software benötigt man vor allem Wissen[4]. Es scheint darauf anzukommen, innerhalb von 12 bis 18 Monaten einen Marktanteil von mehr als 40 % zu erreichen, damit man allein durch Mundpropaganda Marktführer wird[5]. Damit steigt wiederum die Rendite[6].

2. ist ein komplexes Produkt

a) umfasst viele Zeilen Quelltext

Anspruchsvolle Software umfasst typischerweise viele Zeilen Quelltext. Die Metrikgrösse Lines of Code[7] (LOC) ist mit Vorsicht zu betrachten, da es in der Informatik allgemein als vorzugswürdig angesehen wird, unter Beibehaltung des Funktionsumfanges gerade so viele Zeilen zu produzieren, wie eben notwendig und damit Redundanzen zu vermeiden. Ferner muss man das Einsatzgebiet der Software unterscheiden: Eine technische/mathematische Anwendung umfasst häufig nur wenige hundert Zeilen, während bei gleichem Aufwand mehrere hunderttausend Zeilen betriebswirtschaftlicher Anwendung realisierbar wären. Pro Ingenieurmonat kann man von 350 Zeilen ausgehen[8]. Dennoch stellt sich empirisch heraus, dass ein kleines Projekt durchaus 50.000 Zeilen umfasst, während grosse Projekte bis hin zu 30 Millionen Zeilen umfassen[9]. Nun geht es hier nicht um den Schreibaufwand, sondern um die Komplexität, die derart grosse Werke in sich tragen: Schliesslich müssen die Zeilen zueinander passen und ein stimmiges

1. Vgl. Hoch/Roeding/Purkert/Lindner S. 38.
2. Vgl. Hoch/Roeding/Purkert/Lindner S. 38.
3. Vgl. Hoch/Roeding/Purkert/Lindner S. 39.
4. Vgl. das Zitat von Liemandt, Hoch/Roeding/Purkert/Lindner S. 38.
5. Vgl. Hoch/Roeding/Purkert/Lindner S. 41.
6. Vgl. Hoch/Roeding/Purkert/Lindner S. 41.
7. Vgl. Balzert S. 59 f.
8. Ohne Kommentare, die Zahl basiert auf einer Studie der Firma Hewlett-Packard; vgl. Balzert S. 60. Das erscheint sehr niedrig, doch ist der Quelltext lediglich das Endergebnis eines meist komplexen Erstellungsprozesses. Detaillierter Thaller S. 121-126, der weitere Beispiele nennt und die Parameter näher erläutert. Wenn ein Entwickler genau weiss, was er tun muss, und keine Verwaltungs- und Dokumentationsaufgaben hat, können es auch über 1000 Zeilen pro Tag sein.

„Gesamtkunstwerk“ ergeben.

b) ist oft komplex

Anspruchsvolle Software umfasst typischerweise viele Function Points[1]. Diese Metrikgrösse wird gerne an Stelle der Zeilenzahl (LOC) benutzt. Ein Function Point beschreibt eine bestimmte Funktionalität, die das Programm bieten soll. Die Function Points werden in drei Schwierigkeitsklassen eingeteilt. Aus der Vergangenheit kennen die Hersteller den Zeitbedarf für die jeweiligen Schwierigkeitsklassen von Function Points, so dass man neue Projekte vom Zeitaufwand her abschätzen kann. Ein Function Point ist sprachunabhängig, der Zeitbedarf zu seiner Realisierung jedoch u.a. sprachabhängig. Insofern setzt die Anwendung der Function Point-Metrik voraus, dass man bereits Erfahrungen mit der verwendeten Entwicklungsumgebung[2] hat. Gerade bei modernen Projekten ist das selten der Fall. Dann ist die Komplexität bzw. der Aufwand zur Realisierung schlecht abschätzbar.

c) lässt sich schlecht evaluieren

Zwei Software-Produkte lassen sich aufgrund ihrer Komplexität schlecht mit einander vergleichen. Auch die absolute Evaluation anhand vorgegebener Kriterienkataloge ist problematisch, da die meisten Nutzer überhaupt keine präzisen Vorstellungen davon haben, welche Funktionalität sie wirklich benötigen. Es gibt auch keine derartigen Tests z.B. in Fachzeitschriften[3]. Vorsichtshalber lizensieren sie dann den Marktführer[4]. Auch mit zeitlich beschränkten Testversionen gibt es Probleme. Wenn man sie z.B. vier Wochen ernsthaft evaluiert hat, ist die Wahrscheinlichkeit gross, dass schon nennenswerte Arbeitsergebnisse damit erstellt wurden, die ggf. nicht in ein anderes Programm übernommen werden können. Die Evaluation ist manchmal teurer als die eigentlichen Lizenzkosten, sofern es sich nur um eine geringe Anzahl von Lizenzen handelt. Schliesslich muss man die Zeit[5] für die Eva-

9. Vgl. Thaller, S. 37: Ein Textverarbeitungsprogramm in der Programmiersprache ADA kommt auf 38.732 Zeilen, Steuerungssoftware für ein Stahlwalzwerk auf 500.000 Zeilen, Steuerung eines Kernkraftwerks auf 1.500.000 Zeilen, gesamte Software einer Boeing 777 etwa 4.000.000, Windows NT auf 4.300.000, Windows NT 5.0 auf 30.000.000, die Software der US-Steuerbehörde IRS mit 19.000 separaten Applikationen auf insgesamt 62.000.000 Zeilen. Linux kommt auf 3.700.000 Zeilen, vgl. http://www.heise.de/newsticker/data/odi-24.08.01-001.
1. Vgl. „Die Function Point-Methode“ bei Balzert, S. 73-88, Kurzdarstellung S. 77.
2. Im weiteren Sinne: Programmiersprache, Anwendungsdomäne, Plattform, Tools.
3. Die Autoren geben lediglich wieder, warum die Software ihnen gefallen hat, aber prüfen keine absoluten und vollständigen Kriterienkataloge durch. Die Evaluationen ähneln eher Kunstkritiken als der Analyse eines technischen Systems.
4. Vgl. Hoch/Roeding/Purkert/Lindner S. 41.
5. Für Einarbeitung, Erlernen des Programmes und Erstellen der Daten.

luation ebenfalls in die Berechnung einbeziehen.

d) lässt sich schlecht simulieren

Der Aufwand zur Erstellung von Prototypen[1] ist fast genauso hoch wie für die die Erstellung der Endversion. Das verwundert zunächst, denn es ist deutlich einfacher und kostengünstiger, ein Gipsmodell von einem Haus zu erstellen, als das Haus selbst zu bauen. Auch die Statik lässt sich vergleichsweise standardisiert und mit wenigen Parametern durchrechnen.

aa) Skalierung problematisch

In einem Software-Prototypen müssen bereits alle wesentlichen Aspekte abgedeckt werden. Während das Hausmodell keinen festen Bezugspunkt benötigt, hat Software in der Regel eine bestimmte Hardware als Referenzgrösse. Diese Hardware kann man nicht einfach um den Faktor 1000 (z.B. bezüglich der Geschwindigkeit oder Speicherausstattung) skalieren. Auch die Datenmenge kann man nicht um den Faktor 1000 reduzieren, da die meisten Algorithmen eine nichtlineare Komplexität besitzen[2]. Hinzu kommt, dass sehr viele Anforderungen an Software überhaupt nicht skalierbar sind[3].

bb) Schwerpunkt auf Korrektheit

Ein Schwerpunkt bei Software liegt auf der Korrektheit. Diese ist nur in Ausnahmefällen abhängig von der Menge der zu verarbeitenden Daten und lässt sich in der Regel nur mit „korrekt“ oder „fehlerhaft“ bewerten (weil der Benutzer ein nur beinahe richtiges Berechnungsergebnis nicht akzeptiert). Allenfalls die Anzahl der Fehler ist noch zählbar und ihr Schweregrad für die Nutzbarkeit der Anwendung bestimmbar. Die meisten Fehler führen aber direkt zur Nichtbenutzbarkeit durch den Anwender bzw. zur Verweigerung der Abnahme.

cc) Schwerpunkt auf Problemverständnis

Ein weiterer Schwerpunkt bei der Erstellung von Software liegt auf dem Verständnis des Problems. Wenn man von trivialen Aufgabenstellungen absieht, nimmt der Aufwand für das Verstehen der genauen fachli-

1. Ein Prototyp dient zum Überprüfen der Machbarkeit und enthält daher bereits die wesentliche Funktionalität, sozusagen im „Rohbau“. Modelle hingegen sind nur schemahafte Reduktionen des Programmes, meist in grafischer Notation, z.B. mit Hilfe der Sprache UML, die nicht ausgeführt werden können.
2. Bei doppelt so vielen Daten dauert es *mehr* als doppelt so lange, und dieses Mehr steigt mit grösserer Datenmenge nicht nur absolut, sondern auch *prozentual*. Bei ungeschickt gewählten Algorithmen kann dies zum „Stillstand“ des Programmes führen. Beispiel: 100 Datensätze brauchen 1 Sekunde zur Verarbeitung, 1000 Sätze über 2 Stunden, 10.000 Sätze etwa 11 Tage und 100.000 Sätze über 3 Jahre.
3. So ist die Anforderung, dass in einem bestimmten Fall ein „Wirklich löschen?“-Dialog erscheint, nicht skalierbar. Entweder der Dialog erscheint oder nicht.

chen Anforderungen, mitsamt aller Regeln und Ausnahmen, häufig einen grossen Teil der Zeit ein[1].

dd) Zusammenfassung

Wenn man den Prototypen einer Software erst einmal hat, ist es nicht mehr weit bis zur Endversion, da eine Skalierung – wenn überhaupt – allein durch die Hardware vorgenommen wird. Im Idealfall fehlt nur das Installationsprogramm und die Benutzerdokumentation.

e) muss oft speziell angepasst werden

Softwaresysteme werden auf dem Markt häufig als „Fertighaus“ angeboten[2]. Der Nutzer wird, wie auch in der Haus-Analogie, typischerweise eine Reihe von Anpassungen wünschen[3]. Der Anpassungsbedarf geht auf die Umgebung zurück, in der die Software eingesetzt werden soll. Bei betriebswirtschaftlicher Software sind nicht nur Kostenstellen oder Warengruppen[4] einzurichten, sondern auch die individuelle Logistik[5] anzupassen und Altsysteme[6] einzubinden. Bei technischer Software müssen Branchenanpassungen vorgenommen werden[7]. Bei Systemsoftware müssen Treiber für individuelle Geräte[8] installiert oder sogar erstellt werden.

Ein typisches Beispiel ist die Software von SAP. Wesentliche Komponenten sind standardisiert. Jedoch ist ein nicht unwesentlicher Anpassungsbedarf vorhanden, sofern die Software an die Abläufe im Unternehmen angepasst werden soll und nicht umgekehrt[9]. Solche Anpassungsprojekte liegen nicht selten im Bereich mehrerer Millionen Euro. Das macht deutlich, dass selbst bei Standardkomponenten noch ein hoher Aufwand entsteht.

f) ist schwer bis zur Marktreife zu entwickeln

Es ist schwer, Software so zu entwickeln, dass sie reif für eine Verbrei-

1. Ein gutes Beispiel ist eine Software zur Berechnung der Einkommenssteuer. Dazu muss der Entwickler dem Fachexperten entlocken, wie die entsprechende Berechnung fachlich aussieht. Vergisst der Experte auch nur einen seltenen Sonderfall, so wird die Software weitgehend wertlos. Dieser muss dann häufig mit grösserem Aufwand nachträglich eingebaut werden.
2. Beispiele siehe http://www.sap.de, http://www.intershop.de.
3. Vgl. BMBF S. 95.
4. Es kann feste Warengruppen geben (Einzelhandel) oder auch extrem konfigurierbare Artikel (Autos), für die eine Einteilung in Warengruppen keinen Sinn macht.
5. Die Art der Versendung der Waren. Die Ware kann beispielsweise per Spedition verschickt werden, per Post, per Kurier oder auch vom Kunden abgeholt werden. Vgl. zu Logistikprozessen: Scheer, Teil B.
6. Typischerweise existieren bereits Spezialsysteme, die auf einem Alleinstellungsmerkmal des Unternehmens beruhen oder nur aufwändig zu ersetzen wären.
7. Eine CAD-Software muss z.B. für den Automobilbereich oder Architekturbereich angepasst werden.
8. Beispielsweise um besondere Massenspeicher-Subsysteme oder Archivierungslösungen überhaupt ansprechen zu können. Oder um die Geschwindigkeit der Zugriffe zu erhöhen (Tuning).

tung am Markt ist. Das liegt daran, dass die Marktreife vier Dimensionen hat: Funktionstreue (Übereinstimmung der definierten Anforderungen mit dem fertigen Produkt), Qualitätstreue (Übereinstimmung der Qualitätsanforderungen), Termintreue (Einhaltung der dem externen Kunden oder dem Marketing zugesagten Fertigstellungstermins) und Kostentreue (Einhaltung des Personal- und Sachaufwands für die Produkterstellung und -pflege)[1]. Während der Entwicklung ändern sich fortwährend die Anforderungen[2], die Entwicklungs- und Ablaufumgebung (Hardware/Software) sowie die Produktionsmittel (Methoden und Werkzeuge)[3].

g) erfordert Beratung und Schulung

Die meisten Softwareprodukte sind ohne Beratung oder Schulung für den Durchschnittsanwender nicht produktiv einsetzbar. Dabei geht es nicht um die früher vorhandenen „versteckten Funktionen“[4], sondern um die fachliche Nutzung. Die Nutzer möchten mit der Software Mehrwert erzielen. Dazu benötigen sie Knowhow, wie man ein Problem mit der Software effizient lösen kann. Ein Teil der Benutzer erwirbt das Knowhow autodidaktisch, jedoch ist dabei Literatur förderlich[5].

h) Zusammenfassung

Software ist in der Regel ein komplexes, umfangreiches Werk, das schwer bis zur Marktreife zu entwickeln ist, beim Anwender Schulungen und Beratung erfordert und oft angepasst werden muss.

3. ist rechtlich geschützt

a) ist urheberrechtlich geschützt

Das deutsche Urheberrecht schützt auch Software. Dies ist der primäre rechtliche Schutz für Computerprogramme[6]. Er umfasst nach § 69 a I

9. Gerade bei SAP-Einführungen wird dieser Anpassungsbedarf häufig unterschätzt. Das Marketingargument „Es ist schon alles vorhanden“ sagt eben noch nichts darüber aus, in welcher Form die Funktionalitäten realisiert sind. In vielen Unternehmen ist das Phänomen, dass die vorherige Einführung von Individualsoftware an den individuellen Abläufen des Unternehmens gescheitert ist, man auf SAP umgestellt hat (Argument: ist günstiger) und plötzlich angesichts der Anpassungskosten zum ersten Mal sichtbar wird, wie teuer die Abweichung von den SAP-Vorgaben ist. In der Folge strukturieren viele Unternehmen doch ihre eigenen Abläufe so um, wie SAP es annimmt. Die Softwarekosten erzeugen also einen Anpassungsdruck auf die betrieblichen Abläufe.
1. Vgl. die Darstellung bei Balzert S. 34 f.
2. Z.B. weil der Gesetzgeber das Steuerrecht ändert.
3. Vgl. Weinberg, S. 25 „What makes a Good Program?“.
4. Ursprünglich gab es keine Menüs, sondern nur Tastenkombinationen. Diese waren nur schwer zu überblicken und änderten sich ggf. je nach Kontext. Ein gutes Beispiel dafür ist der Editor vi, mit so einsichtigen Tastenkombinationen wie „Escape Doppelpunkt w q“ für „Speichern und Programm verlassen“. Heute ist die Interaktion mit dem Benutzer intuitiver.
5. Es gibt auch Benutzer, die Software ohne Literatur, Schulungen o.ä. selbst erkunden. Diese Fähigkeit ist förderlich in Bereichen, in denen sich die Software so schnell ändert, dass Bücher nicht rechtzeitig verfügbar sind und schnell veralten.

UrhG auch das Entwurfsmaterial. Die ihm zugrunde liegenden Ideen und Grundsätze sind gemäss § 69 a II 2 UrhG nicht geschützt. Die Software muss ferner laut § 69 a III 1 UrhG das Ergebnis einer eigenen geistigen Schöpfung ihres Urhebers sein, jedoch ohne Berücksichtigung qualitativer oder ästhetischer Aspekte (S. 2). Damit scheiden rein automatisch, von Maschinen generierte Programme aus[1]. Nach § 69 c UrhG darf Software nicht ohne Zustimmung des Urhebers kopiert werden.

b) ist in ihrer Interoperabilität urheberrechtlich geschützt

Über § 69 d III UrhG ist es auch ohne Zustimmung des Urhebers erlaubt, das Funktionieren eines Programmes zu beobachten, zu untersuchen und zu testen (Reverse-Engineering[2]). Dadurch soll sicher gestellt werden, dass die Interoperabilität mit anderen Programmen hergestellt werden kann. Man darf z.B. die gespeicherten Dateien für verschiedene Eingaben untersuchen, um das Dateiformat zu analysieren[3]. Auch Speicherauszüge oder der Einsatz eines Debuggers zum zeilenweisen Analysieren der Signalkommunikation ist möglich[4]. Unter bestimmten Voraussetzungen ist sogar über § 69 e UrhG eine Dekompilierung erlaubt. Damit wird eine Öffnung für den Wettbewerb bezweckt[5]. Der Urheber kann diese Massnahmen nur verhindern, wenn er die Informationen selbst kostenlos allgemein verfügbar macht[6]. Ein vertraglicher Ausschluss der §§ 69 d III und e ist gemäss § 69 g II nicht möglich[7].

c) unterliegt dem Urhebervertragsrecht

Die Neufassung des Urhebervertragsrechts durch das „Gesetz zur Stärkung der vertraglichen Stellung von Urhebern und ausübenden Künstlern“[8] vom 22.03.2002 gilt auch für Software. Gemäss § 32 I kann der Urheber danach die Anpassung des Vertrages verlangen, wenn sich herausstellt, dass die vereinbarte Vergütung zum Zeitpunkt des Vertragsschlusses nicht angemessen gewesen ist. § 32 III verhindert das Abbedingen oder Umgehen der Regelung. Open-Source-Lizenzen dürften

6. Vgl. Schricker/Loewenheim vor §§ 69 a ff. Rn 7.

1. Z.B. die von John R. Koza in seinem Buch „Genetic Programming“, 1992, vorgestellten Programme. Eine Software generiert mehr oder minder zufällige Bitfolgen, die als Programme ausgewertet werden. Dann testet die Software, ob das gewünschte Ergebnis durch eines der generierten Programme erreicht wird, ändert sie ggf. automatisch ab und startet einen neuen Evolutions-Zyklus. Dabei erfolgt kein direkter Eingriff durch Menschen, vgl. auch Möhring/Nicolini/Hoeren § 69 a Rn 14.
2. Vgl. zum Begriff: Schricker/Loewenheim § 69 e Rn 4-6.
3. Vgl. Schricker/Loewenheim § 69 d Rn 21.
4. Vgl. Schricker/Loewenheim § 69 d Rn 21.
5. Vgl. Schricker/Loewenheim § 69 e Rn 1.
6. Vgl. Schricker/Loewenheim § 69 e Rn 15.
7. Vgl. Schricker/Loewenheim § 69 g II.
8. Vgl. http://www.urheberrecht.org/UrhGE-2000/download/bgbl102021s1155.pdf.

davon jedoch regelmässig nicht betroffen sein, da sie in Übereinstimmung mit § 32 III 3 UrhG regelmässig jedermann unentgeltlich ein einfaches Nutzungsrecht einräumen und damit der Anspruch auf angemessene Vergütung entfällt[1]. Kritisch könnte das nur dann sein, wenn die GPL unwirksam wäre und der Urheber ggf. einen über Lizenzanalogie zu berechnenden Schadenersatzanspruch gegen den Nutzer hätte.

d) ist in Spezialfällen durch Patentrecht geschützt

Dazu muss die Software neu sein, eine gewisse Schöpfungshöhe aufweisen und gewerblich nutzbar sein. Ferner kommt es auf ihren Technikbezug an, da „Software als solche" gesetzlich nicht patentierbar ist[2]. Patentierbar sollen Ansprüche sein, die zur Lösung eines Problems der Ingenieur- oder Naturwissenschaften die Abarbeitung bestimmter Verfahrensabschnitte durch einen Computer vorschlagen[3]. Sonst ist zu prüfen, ob die auf Datenverarbeitung mittels eines Computers gerichtete Lehre sich durch eine Eigenheit auszeichnet, die unter Berücksichtigung der Zielsetzung patentrechtlichen Schutzes eine Patentierbarkeit rechtfertigt[4]. Es läuft also darauf hinaus, dass die Software neben der reinen Ansteuerung des Rechners noch weitere physikalische Wirkungen verursachen muss. Bekannte Beispiele für die Patentierbarkeit sind Software für eine Tauchuhr oder ein Antiblockiersystem, für Nichtpatentierbarkeit eine Textverarbeitung oder ein Warenwirtschaftssystem[5]. In den USA sind hingegen auch Geschäftsmethoden patentierbar, d.h. Verfahren ohne technischen Bezug. Sollte dies bei der Anpassung des Rechts in Europa übernommen werden, befürchtet man eine Innovationshemmung, insbesondere bei kleinen und mittelständischen Unternehmen sowie Entwicklern freier Software[6].

e) ist durch UWG geschützt

Auch §§ 1, 17 ff. UWG können zum Schutz von Software führen, z.B. wenn Geschäftsgeheimnisse bzw. Know-How berührt sind[7]. Beispielsweise kann ein Mitarbeiter durch einen Wettbewerber abgeworben wer-

1. Vgl. den ursprünglichen Vorschlag in der BT-Drucksache 14/6433 S. 3: „Auf den Anspruch auf angemessene Vergütung kann im Voraus nicht verzichtet werden, soweit der Urheber nicht jedermann unentgeltlich ein einfaches Nutzungsrecht einräumt." samt Begründung für diese Formulierung bei Jaeger/Metzger, http://www.urheberrecht.org/UrhGE-2000/download/stellungnahmen/urhebervertragsrecht.pdf, S. 3-4.
2. Vgl. zur Rechtslage in Deutschland: BMWi S. 125 ff.
3. Vgl. Rössel S. 91.
4. Vgl. Rössel S. 91.
5. Vgl. Rössel S. 92 mit einer Fallgruppenübersicht.
6. Vgl. Rössel S. 92 m.w.N. zum Kommissionsvorschlag der EU.
7. Vgl. Schricker/Loewenheim § 69 g Rn 1.

den und Software oder einfach das Know-How dazu mitnehmen (unzulässige Leistungsübernahme bzw. Schutz als Betriebsgeheimnis[1]). Der ehemalige Arbeitgeber könnte dann gegen den Wettbewerber vorgehen, sofern dieser aus dem Verhalten des Mitarbeiters (selbst ohne unmittelbare Patent- oder Urheberrechtsverletzung) Vorteile zieht.

f) hat ein geringes Haftungsrisiko

Vertraglich ist die Haftung für Software einschränkbar. Konkret kommt es auf die Ausgestaltung des Vertrages an. Sofern es sich um einen individuell ausgehandelten Vertrag handelt, kann der Hersteller die Haftung für einfache und grobe Fahrlässigkeit abbedingen[2]. Sofern es sich um AGB handelt, kann er höchstens die Haftung für einfache Fahrlässigkeit abbedingen[3]. Es ist jedoch kein Fall bekannt, in dem ein grosser Anbieter von Standardsoftware für Fehler in seinem Produkt hätte haften müssen. Das steht in deutlichem Gegensatz zur Praxis bei Gebrauchsgütern, wo teilweise erhebliche Haftungssummen zu veranschlagen und Rückrufaktionen mit kostenloser Nachbesserung üblich sind[4]. Weder individualvertraglich und erst recht nicht über AGB lässt sich die Haftung für Vorsatz ausschliessen. Im übrigen werden bei Standardsoftware auch Mängelrechte selten geltend gemacht[5].

g) lässt sich leicht kopieren

Ein Schutz durch Normen ist nur so gut, wie seine Implementierung. Software lässt sich in der Regel leicht kopieren, selbst wenn einfache Kopierschutzmassnahmen verwendet wurden. Anleitungen dazu gibt es in nahezu allen Fachzeitschriften[6]. Über Datennetze wie Internet und private Mailboxsysteme können Programme schnell und günstig kopiert werden. Grössere Datenmengen lassen sich auf CD-ROM oder DVD kopieren. Die dazu notwendige spezielle Hardware ist für wenige hundert Euro erhältlich, Medien sind im ein- bis zweistelligen Eurobereich zu haben[7]. Etwa 43 Millionen derartige Rohlinge sollen allein im Jahr 2001 mit Spielesoftware versehen worden sein, im Vergleich zu rund 70 Millionen verkauften Exemplaren[8].

Selbst grosse Konzerne setzen nicht korrekt lizensierte Software ein.

1. Vgl. Schricker/Loewenheim vor §§ 69 a ff. Rn 12-14.
2. Vgl. Zahrnt/Erben S. 5.
3. Vgl. Zahrnt/Erben S. 5.
4. Beispielsweise im Automobilbereich.
5. So ist dem Nutzer z.B. mit einer Wandlung kaum geholfen, da er dadurch immer noch kein funktionsfähiges Programm hat.
6. Vgl. http://www.spiegel.de/netzwelt/netzkultur/0,1518,194943,00.html.
7. Vgl. http://www.alternate.de, CD- und DVD-Brenner.
8. Vgl. http://www.spiegel.de/netzwelt/netzkultur/0,1518,194943,00.html.

Häufig sind einfach nur zu wenig Lizenzen vorhanden. Der Schwerpunkt dürfte aber im privaten Bereich liegen, wo korrekte Lizenzen schon fast die Ausnahme sind. In einer Umfrage gaben 57 % der Nutzer an, selten oder nie für urheberrechtlich geschützte Software zu bezahlen, 12 % verwenden „Raubkopien“[1]. Für den Rechteinhaber gibt es keine praktikable Möglichkeit, die Rechtsverstösse zu bemerken, geschweige denn zu verhindern. An den Arbeitsergebnissen lässt sich nur sehr selten erkennen, welche Software verwendet wurde.

h) könnte in Zukunft durch DRM-Systeme geschützt werden

Digital Rights Management-Systeme versprechen einen wirksamen Schutz gegen jegliche Art unberechtigter Programm- und Datennutzung[2]. Im Kern geht es bei solchen Systemen darum, jegliche Art von Kopiervorgängen zu überwachen. Das DRM-System kann diese protokollieren, gestatten oder ablehnen. In der Regel nutzen die Systeme kryptografische Methoden, um sicherzustellen, dass die Programme bzw. Daten nicht am DRM-System vorbei kopiert werden können[3]. Die Systeme setzen aber eine lückenlose, sichere Verbindung zwischen den Anwendungsprogrammen und der Hardware voraus, um sicherzustellen, dass die Daten nirgends in entschlüsselter Form abgreifbar sind. Zu diesem Zweck wird es in naher Zukunft entsprechende Betriebssystemversionen geben[4]. Bei Festplatten und Monitoren gibt es schon Modelle, die über die Anschlusskabel nur verschlüsselt kommunizieren[5]. Auf breiter Front ist innerhalb der nächsten fünf Jahre nicht mit einer umfassenden Durchsetzung von DRM-Systemen zu rechnen, allein schon weil ein grosser Altbestand an Hardware existiert. Auch wird sich die Neigung der Konsumenten zum Kauf derartiger Systeme in Grenzen halten, da sie das unrechtmässige Benutzen von Programmen und Daten zumindest stark erschweren. Folglich müssten enorme Nachlizensierungen erfolgen. Einige Hersteller und Verbände drängen auf den Einsatz von DRM-Systemen, weil sie dadurch der Pauschalabgabe an die

1. Vgl. http://www.computerwoche.de/index.cfm?pageid=254&artid=36676; die Formulierung kann allerdings auch auf die Verwendung von Open-Source, Shareware usw. hindeuten, denn diese sind zwar urheberrechtlich geschützt, aber erzwingen keine Bezahlung. Die Zahl passt dennoch zum „Napster-Trend“.
2. Vgl. dazu ausführlich: Bechtold S. 2 f. und 23 ff.
3. Bzw. sie zwar kopiert werden können, aber nicht mehr zu entschlüsseln sind.
4. Vgl. http://www.heise.de/newsticker/data/cgl-14.12.01-000.
5. Im Falle von Bild und Ton kann das Signal natürlich per Mikrofon oder Videokamera abgenommen werden, bei Software mit einigem Aufwand vom Prozessor. Allerdings werden für den Menschen unsichtbare bzw. unhörbare „Wasserzeichen“ in Bild und Ton eingebettet, die den Lizenznehmer verraten. Diese lassen sich z.Zt. aber relativ einfach entfernen, vgl. Bechtold S. 62 ff.

Verwertungsgesellschaften aus dem Weg gehen wollen[1].

i) Zusammenfassung

In der Theorie ist Software ausreichend rechtlich geschützt. Praktisch ist dieser Schutz äusserst mangelhaft implementiert und somit wenig wirksam. Es bleibt abzuwarten, ob sich DRM-Systeme am Markt durchsetzen können oder gar gesetzlich vorgeschrieben werden.

4. hat viele Vertriebsmöglichkeiten

a) lässt sich „verkaufen“

Software kann in Deutschland „verkauft“ werden. Damit ist gemeint, dass Software wie ein materielles Gut gehandelt werden kann. In Warenhäusern oder bei Versandhändlern kann man Software „kaufen“. Die verwendeten Lizenzen entsprechen allerdings weniger der Vorstellung von einem Kauf. Typischerweise werden dem Nutzer nur die zwingend notwendigen Rechte eingeräumt. Das Verleihen oder Vermieten der Software ist normalerweise ausgeschlossen. Einige Anbieter versuchten sogar, den Gebrauchtverkauf zu unterbinden[2]. Zudem wird regelmässig die Haftung weitestgehend ausgeschlossen. „Reparaturansprüche“ bei fehlerhafter Software werden ebenfalls nicht geltend gemacht. Einweisung, Schulung und ähnliches gibt es bei Standardsoftware eher selten. Selbst gedruckte Handbücher, die den kompletten Funktionsumfang abdecken, sind kaum noch üblich. Man beschränkt sich oft auf dünne Installationshandbücher. Für den Hersteller der Software bedeutet das: Ideale Bedingungen für einen Vertrieb in grossen Mengen. Für den Kunden handelt es sich im Grunde um eine Einbahnstrasse: Er bezahlt, erhält eine standardisierte Ware, hat danach aber keinerlei Ansprüche mehr gegen den Anbieter[3].

b) lässt sich „verschenken“

In der Praxis gibt es keine Probleme beim „Verschenken“ von Software. Damit ist die kostenlose Lizenzeinräumung durch den Hersteller gemeint[4]. Insbesondere, wenn die Programmdistribution über das Internet erfolgt oder das Weitergeben durch die Nutzer erlaubt wird, entstehen dem Hersteller kaum Kosten durch die Verbreitung[5]. Die „Verschenk-

1. Vgl. den Streit von Hewlett-Packard/Bitkom und GEMA bzgl. Pauschalvergütung, http://www.heise.de/newsticker/data/jk-19.04.02-003, sowie andererseits die irritierende Forderung des Bitkom nach Einführung von Softwarepatenten, vgl. http://www.heise.de/newsticker/data/anw-07.05.02-005.
2. Insbesondere bei OEM-Versionen, die zusammen mit einer bestimmten Hardware verkauft wurden.
3. In Verständnis des Kunden, nicht juristisch betrachtet.
4. Vgl. zum Verschenken von Webbrowser- und Server-Software durch Microsoft zum Schaden von Netscape: Hoch/Roeding/Purkert/Lindner S. 157 f.

barkeit“ würde jedoch eingeschränkt, wenn dem Hersteller Kosten pro vergebener Lizenz entstünden[1].

c) bei konstant niedrigen Distributionskosten

Die Herstellkosten für die erste CD von Microsoft Windows 95 betrugen etwa $ 1 Milliarde[2]. Die Kosten der zweiten CD lagen dagegen bei etwa $ 3. Wenn man die Grenzkosten der Distribution von Software betrachten möchte, muss man mehrere Vertriebsformen unterscheiden.

aa) CD-ROM

Der benötigte CD-Brenner kostet wenige hundert Euro. Ob man eine oder Millionen von Kopien herstellt: Die Kosten für die Kopie z.B. einer CD-ROM liegen bei wenigen Euro bis unter einem Euro[3].

bb) Internet

(a) Eigene Website

Wenn es nur darum geht, die Software (kostenlos) für andere zur Verfügung zu stellen, reicht eines der üblichen Hosting-Angebote aus. Der monatliche Betrag für die Site selbst liegt beispielsweise bei 9 Euro, die Transferkosten[4] pro Gigabyte dann bei 15 bis 7 Euro[5]. Für eine typische Softwarekopie dürften die Kosten für den Anbieter bei 0,01-0,15 Euro liegen. Allerdings kann man bei dieser Lösung nicht auf einer vorherigen Bezahlung bestehen, die Dateien sind für jeden frei zugänglich[6].

(b) Shareware-Sites

Eine günstige Möglichkeit, vor dem Abruf der Software durch den Nutzer zu kassieren, stellen Shareware-Sites dar. Dabei trügt der Name insofern, als der Softwareersteller auch Nicht-Shareware dort anbieten kann. Die entsprechenden Sites verlangen eine Bearbeitungspauschale plus Umsatzbeteiligung, übernehmen dafür aber das komplette Inkasso und die Transferkosten. Die Bearbeitungspauschale liegt pro Abruf bei 3 Euro, plus 25 % vom „Verkaufs“-Preis[7].

---

5. Sofern er kostenpflichtige Bibliotheken benutzt, bieten deren Hersteller oft Lizenzen mit unbegrenzter Weitervertriebsmöglichkeit an oder sind austauschbar.
1. Beispielsweise durch Patentgebühren, Versicherung gegen Haftung aus der Lizenz oder Absicherung von Gewährleistungsfällen.
2. Vgl. Hoch/Roeding/Purkert/Lindner S. 39.
3. Grosse Auflagen werden deutlich billiger, z.B. 0,28 Euro pro CD, vgl. http://www.mediabit.de/cd/1_cd_produktion/cd_produktion_preise_teil_1.html.
4. Im Internet bezahlt der Anbieter typischerweise die Transferkosten. Beispiel: Herr X lädt von www.xyz.de eine Datei herunter. Dann zahlt der Betreiber von www.xyz.de an seinen Provider diese Transferkosten, die sich nach der Dateigröße richten. Herr X trägt zudem seine eigenen Zugangsgebühren zum Internet. Die Gesamtgrösse aller abgerufenen Dateien wird monatlich abgerechnet.
5. Bei grösserer Abnahme sinkt der Preis pro GB, alle Preise von http://www.schlund.de, Angebot WebLight NG.
6. Man kann auch mit kennwortgeschützten Verzeichnissen arbeiten. Das ist aber bei grossen Lizensierungsmengen unpraktikabel.
7. Z.B. http://www.shareit.de.

(c) Download-Sites

Wenn es nicht darum geht, vor dem Herunterladen der Software zu kasieren, kommen auch Download-Sites in Frage[1]. Diese bieten dem Hersteller die Möglichkeit, eine Kopie hochzuladen, die dann von den Benutzern kostenlos abgerufen werden kann. Die Download-Sites finanzieren sich über Werbung oder Sponsoren. Eine Reihe von Anbietern ermöglicht so den Abruf der Programme, verschickt aber den Schlüssel zum Freischalten des Programmes erst nach einer (ggf. kostenpflichtigen) Registrierung.

(d) Zusammenfassung

Im Internet gibt es mehrere Möglichkeiten, eine Software zur Verfügung zu stellen. Drei Wege wurden exemplarisch vorgestellt.

cc) Zusammenfassung

Software ist mit konstant niedrigen Kosten distributierbar, sei es offline per CD-ROM oder online im Internet.

d) bei niedrigen Dokumentationkosten

aa) CD-ROM und Internet

Die Software wird praktischerweise so angeboten, dass – wenn überhaupt – auch eine Dokumentation im PDF-Format[2] enthalten ist, die sich der Nutzer selbst ausdrucken kann[3]. Auf diese Weise sind auch bei häufigen Änderungen an der Dokumentation keine Fixkosten durch Druck und Bindung des Handbuchs zu befürchten. Bezüglich der Distributionskosten gilt das gleiche wie für die Software selbst.

bb) Selbst drucken und binden

Bei Individualsoftware und Kleinserien kann der Softwarehersteller auch selbst den Ausdruck per Laserdrucker übernehmen. Zur Bindung kann ein handelsüblicher Ordner dienen. Alternativ kann auch ein Copy-Center damit beauftragt werden. Dieser Weg eignet sich allerdings nur, wenn der Softwareersteller die Dokumentation selbst versenden möchte. Die Kosten dafür und das Risiko für ein eventuelles Inkasso[4] dürften die Herstellungskosten der Dokumentation deutlich übersteigen. Der Aufwand steigt (in einem bestimmten Bereich) mit zunehmender Zahl der abgesetzten Exemplare. Schon bei mehr als zehn Do-

1. Z.B. http://www.download.com.
2. Vgl. http://www.adobe.de/products/acrobat/adobepdf.html.
3. Vgl. zur EDV-Dokumentation generell und gedruckte vs. elektronische Dokumentation: Beckmann, S. 519 ff. und Brandt, S. 571 ff., zu Handbüchern für Softwareanwender: Bergmann/Pötter/Streitz S. 555 ff.
4. Beispielsweise ist die Postnachname problematisch, wenn der Besteller die Sendung wegen Urlaub nicht annimmt. Auch gibt es Missbrauchsfälle.

kumenten pro Tag[1] wird der Softwareersteller einen Mitarbeiter dafür abstellen müssen, um seine eigene Arbeitskraft nicht zu gefährden. Die Kosten dürften in typischen Fällen bei etwa 20 Euro liegen[2].

cc) Book on Demand

Sofern eine gedruckte Dokumentation angeboten werden soll, der Softwareersteller damit aber wenig Arbeit haben möchte, kann er auf „Book on Demand"-Anbieter zurückgreifen. Die Grundidee dabei ist, dass das Dokument als PDF-Datei eingeschickt wird und vom Anbieter daraus eine Vorlage[3] erstellt wird. Das eigentliche Buch wird allerdings erst dann gedruckt, wenn es tatsächlich ein Kunde bestellt. Damit entfällt das Risiko, grosse Mengen aus Wirtschaftlichkeitsgründen drucken zu müssen, die dann bei einer Neuauflage zu Makulatur werden. Die Kosten für ein 200-seitiges Buch im üblichen Format als Paperback belaufen sich auf einmalig 270 Euro. Bei einem Verkaufspreis von 20 Euro erhält der Autor 5,08 Euro (25,4 %), bei 40 Euro Verkaufspreis 15,36 Euro (38,4 %); der Grenzertrag liegt bei etwa 50 %[4]. Dafür übernimmt der Anbieter die gesamte Abwicklung inklusive dem Inkasso. Wie man sieht, ist dieser Weg deutlich teurer als der Direktvertrieb im Internet. Das liegt neben den Druckkosten an dem hohen Grosshandelsrabatt, über den Buchhändler ihre Kosten decken. Bei BOD-Anbietern ist es zumindest unüblich, dem Buch Software z.B. auf einer CD beizufügen, so dass sich der Weg für die reine Softwaredistribution nicht eignet.

dd) Zusammenfassung

Ein Kostenfaktor bei der Lizensierung von Software ist die Dokumentation. Auch hier muss man für eine Grenzkostenbetrachtung mehrere Medien unterscheiden.

e) Zusammenfassung

Software lässt sich „verkaufen" und „verschenken", wobei sowohl die Distributionskosten für die Software als auch die Dokumentation extrem niedrig gehalten werden können.

1. Das entspricht einem geschätzten Zeitaufwand von mindestens einer Stunde pro Tag für das Erstellen und Versenden der Dokumentationen zuzüglich der Überwachung des Inkassos und der Rückläufer/Reklamationen.
2. Kopierkosten, Ordner, Versand, Nachnahmegebühr, Arbeitszeit.
3. Warum diese Vorlage („Master") notwendig ist, ist nicht ganz einleuchtend. Letztlich wird das Dokument direkt auf einem besseren Laserdrucker ausgegeben und danach beschnitten und gebunden. Vermutlich soll damit der Aufwand für die Aufnahme in die Datenbank des Anbieters abgegolten werden.
4. Vgl. http://www.bod.de/produkte/kalkulator.html.

5. erzeugt wirtschaftliche Phänomene

a) kann ihre Entwickler in kurzer Zeit reich machen

Nur selten gelingt es Unternehmen, die kaum älter als 20-30 Jahre sind, ausserhalb der IT-Branche eine derart starke Marktbedeutung zu erlangen. So ist Microsoft das zweitgrösste US-Unternehmen überhaupt, Oracle steht immerhin auf Platz 32[1]. Die SAP ist immerhin der viertgrösste deutsche Konzern[2]. Auch einige hinter den Firmen stehende Personen sind in vergleichsweise kurzer Zeit vermögend geworden[3]. Gemessen am Einsatz kann Software also durchaus sehr profitabel sein. Dem gegenüber stehen aber auch viele Unternehmen und Personen, die durch Software Geld verloren haben[4].

b) neigt zur Förderung von Monopolen

aa) Betriebssysteme

Ein Grossteil der Benutzer von PCs hat zur Zeit ein Betriebssystem der Firma Microsoft installiert. Das kann man damit erklären, dass zahlreiche Hersteller nur Microsoft-Betriebssysteme vorinstalliert anbieten[5]. Lizensiert der Käufer weitere Software, so steigt seine Investition in dieses Betriebssystem[6]. Das liegt daran, dass die neu lizensierte Software unter alternativen Betriebssystemen, z.B. Linux, nicht einfach nutzbar ist. Der Nutzer könnte eine Emulation bzw. eine Alternativ-Implementierung einsetzen, um die Windows-Programme unter Linux laufen zu lassen[7]. Es ist aber wahrscheinlich, dass dadurch Probleme entstehen. Eine übliche Reaktion der Softwarehersteller ist, sämtliche Verantwortung abzulehnen, sobald das als Voraussetzung genannte Re-

1. Vgl. http://www.ftd.de/ub/in/1014399083538.html.
2. Vgl. http://www.ftd.de/ub/in/1014399083286.html.
3. Bill Gates als Gründer von Microsoft ist immerhin seit 8 Jahren mit $ 52,8 Milliarden der reichste Mann der Welt, vgl. http://www.heise.de/newsticker/data/tol-01.03.02-001; Larry Ellison als Gründer von Oracle verfügt über $ 42,1 Milliarden, Microsoft-Mitgründer Paul Allen über $ 24,4 Milliarden und Microsoft-CEO Steve Ballmer über $ 13,2 Milliarden, vgl. http://www.heise.de/newsticker/data/hod-19.03.01-001; SAP-Gründer Hasso Plattner kommt auf 4,7 Milliarden Euro, vgl. http://www.manager-magazin.de/koepfe/reichste/0,2828,182949,00.html; SAP-Gründer Klaus Tschira auf 2,9 Milliarden Euro, Dietmar Hopp auf 2,4 Milliarden Euro, http://www.manager-magazin.de/koepfe/reichste/0,2828,192221,00.html, SAP-Gründer Hans-Werner Hector auf 1,6 Milliarden Euro, vgl. http://www.manager-magazin.de/koepfe/reichste/0,2828,192223,00.html; Sun-Gründer Andreas v. Bechtolsheim auf 1 Milliarde Euro, http://www.manager-magazin.de/koepfe/reichste/0,2828,192225,00.html.
4. Beispielsweise weil die Entwicklung gescheitert ist oder man sich gegen eine etablierte Konkurrenz nicht durchsetzen konnte.
5. Beispiel: Beim Dell Dimension 4400 Office PC gibt es lediglich die Auswahl zwischen XP und XP Professional, vgl. http://www.dell.de.
6. vgl. auch Sester S. 799.
7. Z.B. Wine; die Entwickler selbst sagen, sie sei noch nicht allgemein einsetzbar, vgl. http://www.winehq.com/about. Alternativ kommt auch VMWare Workstation in Frage, vgl. http://www.vmware.com/products/desktop/ws_features.html. Emulatoren tendieren allerdings zur Langsamkeit.

ferenzbetriebssystem verlassen wird[1]. Allerdings gibt es noch deutlich mehr Betriebssysteme neben Linux, für die der Benutzer sich entscheiden könnte und für die keine Emulation verfügbar ist[2]. Da die Softwarehersteller ihr Produkt meist nur für ein Betriebssystem anbieten, muss der Benutzer sich entscheiden, und in den meisten Fällen tendiert er zum Marktführer. Sonst läuft er in technische Probleme hinein[3] und hat zusätzliche Ausgaben für Software und kompatible Hardware.

bb) Office-Produkte

Besonders deutlich sind die Folgen im Bereich Office-Produkte zu sehen. Dort ist der unangefochtene Marktführer Microsoft Office. Weltweit gibt es rund 600 Millionen PCs[4]. Auf einem grossen Teil dieser Rechner ist das Office-Produkt von Microsoft vorhanden. Das liegt daran, dass ein PC nur dann optimal genutzt werden kann, wenn sich die mit ihm erstellten Daten problemlos austauschen lassen. Verbindliche Standards für Office-Dokumente existieren nicht[5]. Das Format der Microsoft-Produkte ist nicht frei öffentlich verfügbar. Drittentwickler können dadurch nur schwer dieses Format einlesen oder schreiben. Damit fällt es Benutzern, die auf den Dokumentenaustausch angewiesen sind, schwer, andere Produkte einzusetzen. Diese Monokultur erleichtert einerseits den Dokumentenaustausch, zementiert aber auch die Tatsache, dass es nur wenige Produkte gibt, die ernsthaft von den Nutzern in die engere Wahl gezogen werden. Die Nutzer verwenden lieber nicht oder nicht ordnungsgemäss lizensierte Produkte, als auf diese Kompatibilität der Daten zu verzichten.

cc) Flash-Animationen im WWW

Auch im Bereich grafischer Animationen im WWW gibt es solche Quasi-Monopole[6]. Die Firma Macromedia hat mit ihrem Tool Flash eine sehr hohe Marktdurchdringung. Während ein grosser Teil der Benutzer Flash bereits als Browser-Plugin installiert hat[7], ist die Unterstützung für offene Standards wie SVG[8] (oder auch Microsofts proprietäres VML) noch sehr gering. Anbieter von Inhalten im Web setzen daher

1. Ein schwächeres Argument als man annehmen mag, denn wenn die Software auf dem Referenzbetriebssystem nicht läuft, wird es auf die Hardware geschoben.
2. Die wesentlichen dürften sein: Solaris, HP-UX, IRIX, AIX, BSDI/OS, DG/UX, FreeBSD, NetBSD, OpenBSD, OS/2, QNX, SCO UNIX, Tru64 und Mac OS X.
3. Vgl. Hoch/Roeding/Purkert/Lindner S. 41.
4. Sicherheit, VBG, vgl. http://www.vbg.de/publikation.
5. Es gibt also keine diesbezüglichen DIN- oder ISO-Standards.
6. Vgl. http://www.heise.de/newsticker/data/ecp-04.05.02-000.
7. Unter anderem durch Vorinstallation seitens der Browser-Hersteller.
8. Vgl. http://www.w3.org/TR/SVG11.

weiterhin auf Flash, da sie eine möglichst grosse Zahl von Nutzern erreichen möchten. Theoretisch ist es zwar auch möglich, parallel zu Flash das SVG-Format zu unterstützen, doch dies verursacht einen höheren Aufwand für den Betreiber ohne unmittelbaren Vorteil.

dd) Zusammenfassung

Der Nutzer ist in vielen Bereichen auf Standardprodukte angewiesen, deren Anbieter häufig eine monopolähnliche Stellung haben[1].

c) Investitionssicherung

aa) für Benutzungs-Knowhow

Ein weiterer Punkt ist das Knowhow im Umgang mit Software[2]. Zwar sind die wesentlichen Funktionen der Produkte sehr ähnlich. Dennoch ist es für den Nutzer angenehmer, eine Umgebung zu nutzen, die er bereits kennt und an die er sich gewöhnt hat[3]. Zudem kann er für solche Produkte leicht kostengünstig Hilfe erhalten[4]. Darüber hinaus gibt es eine grosse Menge Berater, die sich diesbezügliches Knowhow angeeignet haben und ihre eigene Ausbildung refinanzieren müssen[5]. Diese ist oft teuer[6]. Auch das ist ein Grund für die Monokultur der Produkte[7].

bb) für Dritt-Software

Oft werden mehrere Programme eingesetzt, die zusammen arbeiten sollen. Für den Nutzer ist die möglichst reibungslose Zusammenarbeit wichtig. Typischerweise kann er nicht einfach ein Programm gegen ein anderes austauschen, weil die Schnittstellen nicht standardisiert sind. Zwar gibt es eine Reihe von technischen Standards[8], doch reichen meist schon geringe Unterschiede in den fachlichen Schnittstellen[9] aus, um eine erfolgreiche Kopplung zu verhindern. Diese Situation entsteht daraus, dass jeder Hersteller einfach nach Belieben seine eigene Schnitt-

1. So läuft gegen Microsoft in der EU ein Kartellverfahren, vgl. http://www.spiegel.de/wirtschaft/0,1518,195419,00.html.
2. Dazu gehört die Bedienung ebenso wie die Kenntnis von Fehlern einer bestimmten Version und entsprechender Workarounds.
3. Vor wenigen Jahren gab es eine umfangreiche Debatte über die Total Costs of Ownership (TCO). Dabei wurde argumentiert, dass Open Source-Software zwar keine Lizenzgebühren kostet, aber Schulungsaufwand verursachen kann und dadurch insgesamt sogar teurer sein könnte.
4. So bieten z.B. die meisten Volkshochschulen seit Jahren EDV-Kurse für die Office-Produkte von Microsoft an. Für Konkurrenzprodukte sucht man solche Angebote oft vergeblich. Oder einfach durch Kollegen und Freunde.
5. So gibt es z.B. für Oracle und SAP zehntausende Berater, für Intershop rund 5000, vgl. http://www.computerwoche.de/index.cfm?pageid=254&artid=35789; vgl. auch die Stundensätze bei der Freiberufler-Projektbörse http://www.gulp.de.
6. So kostet eine SAP-Zertifizierung für Vertriebspartner etwa 10.000 Euro; vgl. http://www.heise.de/newsticker/data/hps-15.05.02-000.
7. Vgl. Hoch/Roeding/Purkert/Lindner S. 41.
8. Z.B. CORBA, COM/DCOM, .NET, J2EE. Diese legen fest, *wie* über Programmgrenzen hinweg Funktionen aufgerufen und Daten übergeben werden können.
9. Also beispielsweise *welche* Funktionen überhaupt vorhanden sind, *welche* Daten dabei übergeben werden und die *Reihenfolge* in der das geschieht.

stelle baut. Der Hersteller mit der grössten Marktdurchdringung (das ist oft der erste grössere Anbieter) legt dann das Format fest, ohne dass es dadurch verbindlich wird. Die Formate sind in der Regel nicht (zumindest nicht vollständig) öffentlich dokumentiert.

cc) für existierende Daten

Der Nutzer *wird* mit jeder Nutzung des Programmes zunehmend abhängiger von ihm bzw. seinen Nachfolgeversionen, da er Daten erstellt, die persistent gespeichert werden und nur von dem benutzten Programm wieder gelesen werden können. Es ist daher riskant, mit dem „falschen" Programm eine Aufgabe zu beginnen. Man kann die einmal erstellten Daten normalerweise nicht einfach in das neue Programm übernehmen.

dd) bei Entwicklern

Die Entwickler richten sich nach der Lizensierungswahrscheinlichkeit auf der beim Kunden vorhandenen Softwareumgebung, da sie nicht riskieren möchten, eine Software am Markt vorbei zu entwickeln[1]. Dadurch steigt wiederum das Angebot für die Marktführer. So wird der Status Quo fixiert und ein Wechsel immer schwerer[2].

ee) Zusammenfassung

Das Streben nach Investitionssicherung für Know-How, Dritt-Software, Daten und bei Entwicklern fördert monopolähnliche Strukturen.

d) Sicherheitsprobleme

Die Monokultur führt jedoch auch zu schwerwiegenden Sicherheitsproblemen. So können spezielle Schadprogramme[3] sich leichter ausbreiten, weil sie davon ausgehen können, dass eine bestimmte Umgebung vorhanden ist.

e) Zusammenfassung

Die vielfältigen Abhängigkeiten und mangelnden Standardisierungen führen dazu, dass der Benutzer ebenso wie die Entwickler zum Marktführer tendiert. Die Folge daraus sind die Verstärkung und Schaffung weiterer Abhängigkeiten und Sicherheitsprobleme.

6. Zusammenfassung

Software eignet sich aufgrund der geringen Investitionskosten gut für

1. Wer eine neue Software entwickelt, wird sie zunächst für den Windows-Markt entwerfen, da es den grössten Marktanteil hat, und nicht z.B. für AIX.
2. So hat IBM bei seinem Betriebssystem OS/2 sehr damit zu kämpfen gehabt, dass von Dritten kaum spezielle Anwendungssoftware dafür erstellt wurde. IBM hat schliesslich aus der Not heraus eine sehr gute Windows-Emulation eingebaut. Diese führte aber dazu, dass erst recht niemand mehr nativ für OS/2 entwickelte, vgl. http://www.heise.de/newsticker/data/jk-31.03.02-004.
3. Z.B. „Code Red", http://www.heise.de/newsticker/data/pab-20.07.01-000 und „Nimda", http://www.heise.de/newsticker/data/chk-20.09.01-001.

Markteinsteiger. Sie ist ein komplexes Produkt, das rechtlich hauptsächlich durch das Urheberrecht geschützt wird, welches allerdings mangelhaft implementiert ist. Es gibt viele Vertriebsmöglichkeiten für Software („verkaufen"/„verschenken"), z.B. mit geringen Kosten über das Internet. Ferner neigt Software zur Förderung von Monopolen. Ein heikles Thema ist die Investitionssicherung für Know-How, Dritt-Software und Daten. Die Monokultur führt zu Sicherheitsproblemen.

A 2. Software-Entwicklung

1. ist ein komplexer Prozess

Die Erstellung von Software nimmt häufig mehrere Mannjahre in Anspruch. Grössere Projekte dauern hunderte und tausende von Mannjahren. Die Software der US-Steuerbehörden besteht aus etwa 19.000 Einzelprogrammen[1]. Software wie das Betriebssystem Windows NT besteht aus etwa 4.000.000 Zeilen, an denen 200 Entwickler und Tester arbeiten[2]. Danach wäre jede Person für mindestens 20.000 Zeilen zuständig. Man kann dies vergleichen mit der Erstellung eines Romans über rund 66.000 Seiten mit 200 Autoren und Lektoren. Jede Person ist mindestens für etwa 330 Seiten zuständig. Der Roman muss jedoch in kürzester Zeit zusammengestellt werden, da er in wenigen Jahren veraltet[3]. Auch wenn das Werk nur wenige Fehler enthält, kann es vollständig unbrauchbar werden und der Kunde verweigert die Abnahme. Ferner muss der Roman bei Abgabe alle aktuellen Ereignisse der letzten Wochen berücksichtigt haben[4].

2. ist mehr als nur Codierung

Wenn man Software-Entwicklung zumindest in die Phasen Definition, Entwurf, Codierung und Test einteilt, so liegt der Anteil der Codierung lediglich bei 15-35 %[5]. Der Quelltext ist zwar das Endprodukt des Entwicklungsprozesses, hat aber allein für sich genommen, ohne Kenntnis der Dokumente aus den Bereichen Definition, Entwurf und Test, nur eine geringe Bedeutung. Man mag den Quelltext diesbezüglich als das Ergebnis einer Einweg-Hash-Funktion verstehen[6].

3. muss Qualität durch den Prozess sicherstellen

Software befindet sich im Grunde ständig im Zustand der Überarbei-

1. Vgl. Thaller S. 37.
2. Die Zahl stammt von einer alten NT-Version, mittlerweile sind es 30.000.000 Zeilen und wohl entsprechend mehr Beteiligte.
3. Vgl. zur Alterung von Software: Balzert, S. 26 f.; Halbwertszeit ca. 18 Monate.
4. Vgl. Hoch/Roeding/Purkert/Lindner S. 94 ff., danach wurden allein im Jahr 1995 rund 80.000 Projekte in den USA bei Kosten von rund $ 81 Mrd. abgebrochen.
5. Bei Bertelsmann etwa 15-20 %, bei Hewlett-Packard 34 %, vgl. Balzert S. 68.

tung. Daher ist niemandem damit geholfen, wenn das Endprodukt des Softwareentwicklungs-Prozesses – quasi zufällig – eine hohe Qualität aufweist[1]. Denn diese wird eher zufällig sein, so lange der Prozess an sich nicht eine hohe Qualität sicherstellt. Die Norm DIN EN ISO 9001 wird als weitgehend untauglich angesehen, weil sie nicht auf spezifische Probleme der Softwareerstellung zugeschnitten ist[2]. Passender ist das Capability Maturity Model (CMM)[3]. Die Kosten erhöhen sich dadurch pro Entwicklerjahr z.B. um etwa $ 1375, während die Produktivität um 35 % steigt und die Restfehlerrate um 39 % sinkt[4]. In den USA kommen 73 %, in Europa immerhin 47,9 % aller Unternehmen nicht über Level 1 hinaus[5]. In den USA kommen nur 10 % auf Level 3, 0,6 % auf Level 4 und 0,3 % auf Level 5[6], in Europa erreicht kein einziges Unternehmen diese Level[7].

4. ist abhängig vom Team

Softwareentwicklung ist in hohem Masse abhängig von den beteiligten Personen[8]. Dabei kommt es besonders auf Team- und Kommunikationsfähigkeit an[9]. Wenn das Verhältnis unter den Entwicklern und zum Kunden hin gut ist, gelingen grossartige Werke. Hingegen kann es auch passieren, dass trotz aller materiellen Voraussetzungen keine brauchbare Software entsteht. Einige Autoren sind der Meinung, dass Programmieren eher eine Kunst sei, denn eine Wissenschaft[10]. Denn mit rein technischen Beschreibungen lässt sich nur schwer erklären, was „schöne“ Programme von „uneleganten“ unterscheidet. Der Nutzer oder andere Entwickler merken dies jedoch umso schneller und ohne Bemü-

6. Man kann aus dem Quelltext keinen eindeutigen Schluss auf die Dokumente der Bereiche Definition, Entwurf und Test ziehen, da nur ein Bruchteil der zur Entwicklung benötigten Informationen in diesem Extrakt vorhanden sind. Der Quelltext sagt nur *wie* es ist, aber nicht *warum* es so ist. Das *Warum* ist aber zwingend zur effizienten Weiterentwicklung notwendig.

1. Vgl. den Abschnitt 1.2 „Software-Mängel: Der Regelfall?“ bei Thaller, S. 15 ff., der instruktiv darlegt, welche Kleinigkeiten zu Software-Katastrophen führten.
2. Vgl. Thaller S. 65 und 78. Auf S. 174 finden sich die qualitativen Unterschiede.
3. Vgl. Carnegie, S. 15-20 oder die Kurzdarstellung bei Thaller S. 71 f.
4. Vgl. Thaller S. 211
5. Level 1 bedeutet: „Es gibt wenige stabile Prozesse, und falls sie auf dem Papier existieren, werden sie nicht eingesetzt. Die vorherrschende Devise ist: ’Machen Sie es einfach!’. Der Erfolg basiert auf den heroischen Anstrengungen einzelner Mitarbeiter. Krisenbewältigung (Fire Fighting) ist eine immer wieder auftretende Tägigkeit. Die Beziehungen zwischen verschiedenen Gruppen sind unkoordiniert, manchmal sogar feindselig. Es ist riskant neue Werkzeuge einzuführen, denn es könnte alles zusammenbrechen.“, Thaller, S. 71; zur Statistik: S. 78.
6. Die NASA hat Level 5 erreicht, vgl. Thaller S. 216 f.
7. Vgl. Thaller S. 78.
8. Vgl. Weinberg, S. 67 ff. „The Programming Team“, Zusammenfassung S. 91 f.
9. Vgl. BMBF S. 121 f.
10. So heisst ein Standardwerk der Informatik von Donald E. Knuth „The Art of Computer Programming“, vgl. http://www.heise.de/newsticker/data/hes-17.05.02-000 und http://www-cs-faculty.stanford.edu/~knuth/taocp.html.

hung wissenschaftlicher Kriterien[1].

5. kann blockiert werden

Nicht selten wird die Entwicklung von Software aus Reihen des Auftraggebers blockiert. Das hängt damit zusammen, dass man glaubt, selbst arbeitslos zu werden, weil man sich durch Mithilfe bei der Automatisierung selbst entbehrlich macht oder schlicht fürchtet einen stupideren Job zu haben. Expertensysteme[2] haben auf breiter Front damit zu kämpfen gehabt. Besonders gut kann man die Softwareentwicklung damit torpedieren, dass man Geschäftsprozesse viel komplizierter darstellt als sie eigentlich sind (oder sein müssten). Oder man verlangt geradezu Wunder von der Software, z.B. dass sie automatisch Dinge entscheiden soll, für die keine einheitlichen Entscheidungskriterien angegeben werden (können). Die Anforderungen werden so immer schwieriger zu erfüllen, und die Gefahr des Scheiterns des Projektes steigt überproportional. Selbst wenn Software bereits erstellt ist, kann sie nach der Einführung auf Akzeptanz-Schwierigkeiten stossen[3].

6. wird eher selten nach formalen Methoden durchgeführt

Zur Erstellung von Software können verschiedene Vorgehensmodelle verwendet werden[4]. Grundsätzlich sind diese lediglich eine Absprache unter den beteiligten Personen, welche Schritte man in welcher Reihenfolge vornehmen möchte. Formale Vorgehensmodelle haben einen grossen Einfluss auf die Qualität des Produktes, da ihre Befolgung die Konsequenzen einer chaotischen Entwicklung (Probleme mit Qualität, Kosten und Terminen) verhindert[5]. Zu den anspruchsvolleren Methoden gehören das Wasserfallmodell[6], V-Modell[7] oder der Rational Unified Process[8]. Ein eher lockeres Vorgehensmodell ist in den letzten Jahren unter dem Namen Extreme Programming[9] bekannt geworden. Etwa die Hälfte aller Unternehmen folgen keinem definierten Vorgehensmodell. Weitere 34 % geben an, „unternehmenseigene Vorgehensmodel-

1. Auch das passt zur obigen Roman-Analogie.
2. Darunter versteht man Software, die dem Benutzer im Idealfall das komplette Wissen eines Fachexperten zur Verfügung stellt. Insbesondere kann ein Experte neues Wissen einpflegen und das System erklärt dem Anfrager auf Wunsch seinen Lösungsweg, vgl. Rechenberg/Pomberger/Dorn/Gottlob S. 979.
3. Vgl. Zahrnt, S. 5, Fn 6.
4. Vgl. BMBF S. 127 ff.
5. In der Regel werden die Probleme nicht oder zu spät erkannt, um noch gegensteuern zu können. Vorgehensmodelle legen daher sehr viel Wert auf die Einhaltung vorgegebener Abläufe und deren Dokumentation. Dem V-Modell ist z.B. angesichts der Papierflut vorgeworfen worden, eine Software-Bürokratie zu betreiben.
6. Vgl. Thaller S. 106 ff. oder Jacobson/Booch/Rumbaugh S. 90.
7. Vgl. den Abschnitt „Was ist das V-Modell?“ bei Dröschel/Wiemers S. 1-21.
8. Vgl. Jacobson/Booch/Rumbaugh, S. 3-13.
9. Vgl. http://www.extremeprogramming.org.

le“ einzusetzen[1]. Diese Formulierung deutet darauf hin, dass gar kein wohldefiniertes Modell (im Sinne der oben genannten) eingesetzt wird, da die Entwicklung und Befolgung eines durchdachten Vorgehensmodelles, das positiven Einfluss auf die Qualität hat, typischerweise sehr aufwendig ist, Resourcen bindet und erst mittelfristig positive Auswirkungen zeigt.

7. hat einen grossen Forschunganteil

Softwareentwickler sind bei Neuentwicklungen zu einem grossen Prozentsatz ihrer Arbeitszeit damit beschäftigt, Forschung zu betreiben[2]. Das hat damit zu tun, dass ständig neue Entwicklungswerkzeuge eingesetzt werden sollen[3]. Diese sind typischerweise nicht ausgetestet, spärlich dokumentiert und erst recht nicht ausgereift. Daher ist eine wesentliche Tätigkeit der Entwickler, sich in diese Technologien einzuarbeiten und sie trotz aller Widrigkeiten einsetzbar zu machen[4]. Faktenwissen veraltet sehr schnell[5], das methodische Wissen[6] ist beständiger. Während der gesamten Produktentwicklungszeit verändert sich die Umgebung[7] stark weiter, für die die Software gedacht ist. In der Praxis gelingt es nicht einmal, diese für ein Jahr konstant zu halten. In grossen Projekten ändert sich die Umgebung des Softwareentwicklers mindestens wöchentlich[8].

8. Zusammenfassung

Software-Entwicklung ist ein komplexer Prozess, der weit mehr als nur die Codierung umfasst. Die Qualität der Software kann nur durch den Prozess selbst sicher gestellt werden. Die Entwicklung ist in hohem Maße von den beteiligten Personen abhängig. Sie erfolgt selten nach formalen Methoden und hat einen hohen Forschungsanteil.

1. Vgl. BMBF S. 128.
2. Microsoft gab 1997 $ 1,9 Mrd. für Forschung und Entwicklung aus. Allein für Windows95 waren es schätzungsweise $ 500 Mio., vgl. Hoch/Roeding/Purkert/ Lindner S. 123.
3. Man verspricht sich davon Produktivitätsvorteile. Zum Teil liegt in der Verwendung dieser Technologien auch gerade der Sinn der Neuentwicklung, z.B. bei Internetanwendungen oder Mobiltelefon-Anwendungen.
4. Die Aufgabe ist häufig, aus fehlerhaften und nicht vollständig verstandenen Teilen ein trotzdem funktionierendes Gesamtsystem zusammenzusetzen.
5. Man kann dabei von etwa 18 Monaten ausgehen, in denen sich z.B. in Bereichen moderner Entwicklungsumgebungen die Schnittstellen deutlich ändern oder ganz andere Technologien, z.B. andere Programmiersprachen, verwendet werden.
6. Vorgehensweisen, Lernstrategien.
7. Unter anderem Hardware, Betriebssystem, Schnittstellen, fachliche Umgebung.
8. Beispielsweise Anforderungen oder Schnittstellen zu anderen Subsystemen.

A 3. Urheber entwickeln Software

1. aus wirtschaftlichen Gründen

a) um dadurch Rationalisierungspotentiale zu erschliessen

Das wohl älteste Motiv für die Entwicklung von Software ist die Nutzung von Rationalisierungspotentialen. Ob es sich um die Auswertung einer Volkszählung handelt oder um Budgetierungsarbeiten[1]: Der Computer kann zahlreiche Tätigkeiten schneller und fehlerfreier durchführen als der Mensch. Die Bedingung dafür ist jedoch, dass der Maschine durch das vom Menschen erstellte Programm fehlerfrei und vollständig beschrieben wird, was sie zu tun hat. Ebenso wie ein geniales Programm ein hohes Rationalisierungspotential offenbart, kann ein kleiner Fehler im Programm vernichtende Folgen haben[2].

b) aus strategischen Gründen

aa) Privatpersonen

Bei Privatentwicklern lässt sich beobachten, dass man einem Unternehmen zeigen will, dass es auch „anders“ geht. Das „anders“ kann dabei definiert werden als „billiger“, „unter besseren Lizenzbedingungen“ oder „qualitativ hochwertiger“. Man möchte ein Exempel statuieren. Oder gar einem bestimmten Unternehmen „Schaden“ (weniger Umsatz) zufügen, indem man eine bestimmte Software günstiger auf den Markt bringt bzw. kostenlos anbietet. Das spielt besonders dann eine Rolle, wenn der Softwarehersteller aufgrund seiner Grösse und wirtschaftlichen Stärke z.B. bereits Einfluss auf Parlamente ausübt[3].

bb) Unternehmen

In Unternehmen wird Software nicht selten als Marketinginstrument benutzt. Die Software wird entwickelt, um einem Mitbewerber zu schaden und dadurch selbst eine günstigere Position am Markt zu erhalten[4].

---

1. 1979 entwickelten zwei Studenten die Tabellenkalkulation VisiCalc. Die Budgetierungsarbeiten dauerten damit statt 20 Stunden nur noch 15 Minuten. Ein Buchhalter soll darauf hin vor Aufregung gezittert haben. Innerhalb von 6 Jahren wurden 700.000 Kopien lizensiert. Viele Kunden kauften den Apple-Computer nur wegen dieses Programmes, vgl. Hoch/Roeding/Purkert/Lindner S. 2-4.
2. Kritisch ist dabei nicht der Normalfall, sondern die zahlreichen Sonderfälle, die meist nicht bedacht werden. So gibt es z.B. die Anekdote, dass ein Kampfflugzeug sich bei Überfliegen der Datumsgrenze regelmässig automatisch auf den Kopf drehte. Bei einer Überprüfung der Software stellte sich heraus, dass sich dann ein Vorzeichen änderte. Ein Fall, der so nicht bedacht wurde und auch bei sämtlichen Tests (fernab der Datumsgrenze) nicht aufgefallen war.
3. Vgl. http://www.heise.de/newsticker/data/dwi-09.05.02-000, danach hat die Firma Oracle mutmasslich mehrere Spenden an Politiker verteilt und daraufhin mit dem Bundesstaat Kalifornien einen um $ 41 Millionen überteuerten Lizenzvertrag geschlossen. Microsoft gab im Jahr 1999 $ 3,2 Mio. für politische Lobbyarbeit aus, vgl. http://www.heise.de/newsticker/data/em-15.04.99-000; im Kontext der Linux-Einführung im Bundestag vgl. http://www.heise.de/newstikker/data/wst-16.11.01-003, in Deutschland steht dafür ein siebenstelliger Etat zur Verfügung, vgl. http://www.heise.de/newsticker/data/jes-06.09.01-002.

Beispiel: Die kleine Firma A entwickelt mit hohem Aufwand eine innovative Software. Als die grosse Firma B merkt, dass diese Software zumindest ihrem Image als Innovationsführer schaden könnte, versucht sie, Unternehmen A zu schaden. Mit einem enormen Aufwand entwikkelt B eine Software, die besser ist als die von A und vertreibt sie mit grossem Marketingaufwand günstiger als A. Die Firma A kann ihr Produkt nicht mehr absetzen und erleidet schweren Schaden.

cc) Zusammenfassung

Ein Motiv für die Entwicklung von Software sind politische und strategische Gründe.

c) als Alleinstellungsmerkmal

Ein grosser Teil der von Unternehmen benötigten Software ist in anderen Unternehmen bereits im Einsatz. Jedoch ist diese Software nicht am Markt erhältlich, weil die Software als Alleinstellungsmerkmal gesehen und entsprechend geschützt wird[1]. Der Hersteller der Software wird vielfach verpflichtet, keinesfalls einem Mitbewerber Lizenzen einzuräumen. Oft darf nicht einmal der Quelltext für andere Projekte wiederverwendet werden, da so für Mitbewerber günstigere Softwarelösungen entstehen könnten. Daher sind Wettbewerber in der Regel gezwungen, die für ihren Betrieb zentrale Software selbst entwickeln zu lassen.

d) weil man die Fertigungstiefe erhöhen möchte

Viele Unternehmen haben die Fertigungstiefe verringert (z.B. durch Outsourcing), um sich auf ihre Kernkompetenzen zu konzentrieren und Synergieeffekte zu nutzen. Einigen Unternehmen wird jedoch klar, dass diese Vorteile zu einer erhöhten Abhängigkeit von Zulieferern führen. Die können nun die Preise erhöhen oder ihre Leistung ganz einstellen, wodurch das Unternehmen in ernsthafte Schwierigkeiten kommen kann. Daher werden ausgelagerte Kompetenzen wieder in das Unternehmen zurück verlagert[2]. In diesem Zusammenhang kommt es häufig zur Entwicklung unternehmensspezifischer Software.

e) weil man von Projekten zur Produktentwicklung wechselt

Gelegentlich kommt es vor, dass Unternehmen, die Software im Kundenauftrag entwickelt haben (Projektgeschäft) sich entschliessen, derartige Software als eigenständiges Produkt anzubieten[3]. Eine Variante

4. Vgl. „I fell in love with IBM“, COMPUTERWOCHE Nr. 06 vom 08.02.2002, http://www.computerwoche.de/index.cfm?pageid=267&type=ArtikelDetail&id=80106304&cfid=4796566&cftoken=31071324&nr=1.
1. Vgl. BMBF S. 94.
2. BMBF S. 126.

ist, dass der ehemalige Projektkunde am Erlös des Produktes beteiligt wird. Das ist für ihn nur dann interessant, wenn keine strategischen Gründe dagegen sprechen[1], also beispielsweise wenn die Software[2] für ihn nur eine untergeordnete Rolle spielt. Der Auftragnehmer kann aber auch bei einer Neuerstellung von dem Knowhow profitieren, das er während der ersten Projektdurchführung gesammelt hat. In der Praxis entsteht diese Konstellation häufig dadurch, dass zunächst ein Unternehmensteil ausgegliedert wird, der Software entwickelt. Man möchte diesen nicht nur als eigene Kostenstelle im Unternehmen betrachten, sondern wirtschaftlich vom Hauptunternehmen entkoppeln[3]. Das neue Unternehmen führt in der Anfangszeit die EDV-Projekte des Hauptunternehmens durch. Später stellt es sich auf eigene Beine, versucht auch andere Kunden zu gewinnen und eigene Produkte zu vertreiben[4].

f) weil Mitbewerber Software entwickelt

Selbst wenn ein Unternehmen gar kein aktuelles Interesse an einer bestimmten Produktentwicklung hat: Oft lässt sich nicht absehen, ob sich noch ein Markt dafür entwickelt. Um den Anschluss nicht zu verlieren bzw. den Abstand zu den Mitbewerbern nicht allzu gross werden zu lassen, wird Software entwickelt[5]. Dabei geht es darum, rechtzeitig Erfahrung zu sammeln, um den Aufwand und die Problemstellen ausfindig zu machen[6].

g) um als erster im Markt zu sein

Für innovative (Nicht-Software-)Produkte ist es häufig schwer, eine entsprechende Softwareveredelung anzubieten[7]. Da das Grundprodukt noch nicht auf dem Markt ist, kann auch keine spezielle Software dafür vorhanden sein. Um das Produkt dennoch frühzeitig auf den Markt bringen zu können, ist die Eigenentwicklung einer angepassten Software

3. BMBF S. 131.

1. Beispielsweise wenn sicher gestellt ist, dass auf diese Weise nicht neue Konkurrenten entstehen oder bestehende Konkurrenten gefördert werden.
2. Zumindest in der Betrachtung des Auftraggebers.
3. Dafür kann z.B. eine Tarifbindung des Hauptunternehmens massgebend sein. Die Ausgründung ist dann als eigenes Unternehmen nicht an derartige Verträge gebunden.
4. Die Firma Debis war ursprünglich eine Ausgründung des heutigen Daimler-Chrysler-Konzerns, wickelte dann Projekte auch für andere Auftraggeber ab und gehört heute zu T-Systems.
5. BMBF S. 154.
6. Microsoft hätte den Anschluss fast verpasst, weil man den Konkurrenten Netscape und den Internet-Trend lange Zeit nicht ernst genommen hat, vgl. Hoch/ Roeding/Purkert/Lindner S. 57 ff.
7. Als Beispiel mögen die vor einigen Jahren verbreiteten „Tamagotchis“ dienen. Das waren kleine Spielzeugrechner, die man mit einem Tastendruck „füttern“ musste, wenn sie „vor Hunger“ piepsten. Die Hardware war zwar vorhanden, jedoch war eine entsprechende Software auf dem Markt nicht verfügbar.

notwendig[1].

h) um eine Monokultur zu verhindern

Viele Marktteilnehmer sehen in der monopolartigen Stellung einiger Softwareanbieter ein Problem[2]. Mit Monopolen sind regelmässig auch Abhängigkeiten verbunden, die schlimmstenfalls zur Erpressbarkeit führen können. Es macht daher bei unternehmenskritischer Software auch betriebswirtschaftlich Sinn, rechtzeitig eine Alternative zu beauftragen. Damit vermeidet man eine Monokultur und ihre möglichen negativen Folgen.

i) weil die Eigenentwicklung wirtschaftlicher ist

Wenn Software über einen sehr lange Zeitraum genutzt werden oder auf einer grossen Zahl Arbeitsplätze zur Anwendung kommen soll, kann es wirtschaftlicher sein, sie selbst zu entwickeln und zu pflegen[3]. Dem liegt die Überlegung zu Grunde, dass externe Softwarehersteller nicht nur kostendeckend arbeiten, sondern auch Gewinn machen wollen. Aber auch nicht materielle Werte können ausschlaggebend für die Eigenentwicklung sein. Ein Unternehmen gewinnt an Knowhow, wenn es Software selbst entwickelt. Dieses Knowhow kann es für andere Projekte nutzen oder sogar später anderen Unternehmen anbieten. Hinzu kommt, dass die Loyalität der Entwickler unzweifelhaft ist. Bei externen Auftragnehmern besteht oft das Bestreben, weiterer Aufträge zu generieren. So macht man den Kunden ggf. nicht auf Anforderungen aufmerksam, die er vermutlich hat bzw. entwickeln wird und die später eine teure Überarbeitung zur Folge haben, da man eben an dieser Überarbeitung gut verdienen kann.

j) aus Lizenzgründen

aa) Lizenzgebühr

Jedes Projekt hat einen bestimmten Etat. Dieser definiert letztlich, welche Produkte lizensierbar sind. So war beispielsweise die Quelltext-Lizenz von TopLink[4], einem bekannten objektrelationalem Mapping-

1. BMBF S. 178 („First-Mover-Strategie“).
2. BMBF S. 171 („quasi-monopolartige Stellung weniger Anbieter“).
3. Beispiel: In einer Behörde wird auf 50.000 Arbeitsplätzen eine Textverarbeitung eingesetzt. Der reguläre Preis für Word 2002 als OEM-Version beträgt 220 Euro. Selbst wenn man die Vorgängerversion lizensiert, kostet diese noch 68 Euro. (http://www.softwaretrading.de) Nimmt man an, dass man zur Erledigung der Tätigkeiten eine funktional ausreichende (!) Textverarbeitung – die ggf. weniger Funktionen als Word hat – für 500.000 Euro erstellen lassen kann, so wären das nur 10 Euro pro Arbeitsplatz. Für alle deutschen Behörden ergibt sich ein noch höheres Sparpotential. Zudem ist die Pflege günstiger (pro Jahr 2 Gehälter) und man kennt das Dateiformat. (Das Beispiel scheitert ggf. an Behördenrabatten.) Vgl. die Förderung von Open-Source in Taiwan, http://www.heise.de/newsticker/data/daa-04.06.02-001.

Tool, oberhalb von 10.000 Euro angesiedelt. Für einen Entwickler, der nur überprüfen möchte, ob der Fehler wirklich auf seiner Seite oder in diesem Tool liegt, ist das inakzeptabel. Auch Tools wie Rational Rose oder Together Control Center sind in der Region von mehreren tausend Euro pro Benutzer angesiedelt[1]. Daher entwickelten sich kostenlos erhältliche Programme wie ARGO-UML[2]. Diese decken jedoch nur wenige Anforderungen ab[3]. Im Bereich der EJB-Container sieht es ähnlich aus. Die Anschaffung der beiden Marktführer[4], BEA WebLogic[5] und IBM WebSphere[6], kostet nicht selten fünfstellige Beträge (einmalig), zuzüglich jährliche Wartungskosten[7]. Die Open-Source-Alternative dazu heisst JBoss, und kommt pro Monat auf 100.000 Downloads[8].

Im Bereich der Datenbanken kann man für Oracle[9]-Lizenzen leicht eine sechsstellige Summe ausgeben. Für viele Aufgaben reicht aber MySQL, das etwa 2.000.000 Installationen hat[10].

Auch bei den Betriebssystemen spielen die Lizenzkosten eine Rolle. Während Microsoft Windows XP[11] einen Strassenpreis etwa 230 Euro hat[12], ist Linux kostenlos im Internet erhältlich oder z.B. über die SuSE-Distribution für 49,90 Euro[13].

Problematisch ist insbesondere, dass man Software nur selten z.B. für wenige Monate oder gar Male mieten oder für eine kurze Dauer lizensieren kann[14]. Praktisch existiert kein Markt für gebrauchte Software, so dass man auch dadurch nicht die Kosten reduzieren kann.

bb) Lizenzbedingungen

Viele Open-Source-Produkte stehen unter der GNU Public Licence

4. Siehe http://www.webgain.com/products/toplink.
1. Together Control Center kostet 8970 Euro pro Arbeitsplatz, plus etwa 1000 Euro Support pro Jahr, http://www.togethersoft.com.
2. ArgoUML hat einen kommerziellen Gegenpart, vgl. http://www.gentleware.com.
3. Vgl. zur ersten Übersicht der Produkte: http://www.jeckle.de/umltools.html.
4. Vgl. eine Studie der Meta Group aus dem September 2001 zum nordamerikanischen Markt, zitiert nach http://www.jboss.org; danach kommt BEA WebLogic auf 37 %, IBM WebSphere 22 %, Oracle Application Server 11 %, IPlanet Application Server 5 % und andere auf 12 %. Berücksichtigt wurden nur Umsätze durch Lizenzen, daher sind Open-Source-Produkte nicht enthalten.
5. Preise aus 10/2001: Weblogic Server 6.1 Advantage Edition pro CPU 15.460 Euro, Maintenance 5x8 2.474 Euro; Weblogic Server 6.1 Premium Edition pro CPU (Cluster) 26.290 Euro, Maintenance 5x8 4.206 Euro; vgl. www.bea.com.
6. Siehe http://www.ibm.com.
7. Das umfasst regelmässige Updates.
8. Vgl. http://www.jboss.org.
9. Siehe http://www.oracle.de.
10. Vgl. http://www.mysql.com und http://www.mysql.com/company/index.html.
11. Siehe http://www.microsoft.de.
12. Preis für eine Vollversion unter http://www.softwaretrading.de; für Grosskunden gibt es deutlich günstigere Angebote.
13. Vgl. http://www.suse.de und http://www.edv-buchversand.de/suse/index2.html.
14. Einige Anbieter rechnen dann allerdings sogar pro Benutzung ab oder trauen sich, die Lizenzgebühren an Kosteneinsparungen durch die Software zu koppeln.

(GPL)[1]. Diese erfordert es, dass der Quelltext des Programmes, welches mit einem GPL-Programm gelinkt wurde, auch unter der GPL verbreitet werden muss. Das Ziel dieser Regelung ist, zu verhindern, dass Softwarehersteller zwar GPL-Programme nutzen, aber nichts an die „Open-Source-Community“ zurückgeben.

Für eine Reihe von Projekten ist das inakzeptabel. Das kann daran liegen, dass Softwarehersteller ihren eigenen Quelltext nicht veröffentlichen möchten. Aber auch daran, dass es ihnen durch eine Vereinbarung mit dem Kunden untersagt ist. Das kommt häufig vor, wenn es sich um Geschäftsgeheimnisse handelt, die nicht anderweitig geschützt werden können[2]. In diesen Fällen ist die gewünschte Software-Bibliothek zwar am Markt vorhanden, sie kann aber nicht genutzt werden.

Auch der Haftungsausschluss kann ein wesentliches Kriterium sein. In einigen Unternehmen ist es untersagt, Software einzusetzen, für die kein Anbieter explizit haftet[3].

cc) Zusammenfassung

Es gibt auf dem Softwaremarkt fast alles zu lizensieren, was der Endkunde oder Entwickler benötigt. Das sagt jedoch noch nichts darüber aus, wie hoch der Preis für diese Lizenzen ist und ob die Lizenzen einen akzeptablen Inhalt haben.

k) weil sie gefördert wird

Die Entwicklung von Software wird von einigen grossen Unternehmen stark subventioniert. Das bekannteste Beispiel dafür ist wohl Linux. Die Firma IBM steckte mehrere Milliarden in die Entwicklung von Linux[4]. Auch der Linux-Distributor SuSE sponsort bestimmte Teile von Linux[5]. Auch die Entwicklung von Software anlässlich von Wettbewerben oder Preisauslobungen gehört in diese Kategorie[6].

1. vgl. den Abdruck bei Jaeger/Metzger S. 177 ff.
2. Beispielsweise nicht patentierbar und nur durch §§ 1, 17 UWG geschützt sind.
3. Man denke an den Bank- oder Börsenbereich: Angesichts der grossen möglichen Schadenssummen ist es dort ratsam, die Verantwortung und damit auch die Haftung auf die Softwarehersteller abwälzen zu können und von Lieferanten entsprechende Versicherungsnachweise zu fordern.
4. Vgl. „Der Linux-Kurs des blauen Supertankers“, COMPUTERWOCHE Nr. 17 vom 27.04.2001, http://www.computerwoche.de/index.cfm?pageid=267&type=ArtikelDetail&id=149629&cfid=4796566&cftoken=31071324&nr=20; „I fell in love with IBM“, COMPUTERWOCHE Nr. 06 vom 08.02.2002, http://www.computerwoche.de/index.cfm?pageid=267&type=ArtikelDetail&id=80106304&cfid=4796566&cftoken=31071324&nr=1 vom 28.04.2002.
5. Alsda wäre ISDN4L, Sound- und USB-Unterstützung, http://www.suse.de.
6. Vgl. den Visacard-Wettbewerb zu Smartcards, http://www.heise.de/newsticker/data/cgl-07.06.01-000, und Wettbewerbe des BMWi, z.B. http://www.heise.de/newsticker/data/ad-27.04.02-000.

l) weil einmaliger Bedarf besteht

Anlässlich der Jahr 2000-Umstellung sind viele Software-Lösungen redundant entwickelt worden, um Softwareausfälle zu verhindern. Im Finanzsektor sind in einigen Fällen mehrere gleichartige Ausfallsysteme entwickelt worden, um Softwarekatastrophen zu verhindern, die zur Vernichtung der Unternehmen in kürzester Zeit geführt hätten[1].

Im Rahmen von Umstellungen auf neue Softwaresysteme wird normalerweise eine Migrationssoftware geschrieben, die allein dazu dient, die alten Daten in das neue System zu übernehmen.

m) weil sie ein „grosser Wurf" werden soll

Der Ersteller der Software versucht mit wenig Aufwand eine Software zu erstellen, die dann in den Massenmarkt vordringt und hohe Umsätze bringt. Er geht dabei nach der Überlegung vor, dass es besser ist, eine einfache und ggf. unfertige Software auf den Markt zu bringen, als eine komplexe und sehr robuste Software zu vertreiben. Den Softwareersteller lockt die Idee, dass potentiell Millionen von Kunden in kurzer Zeit seine Software lizensieren könnten. Weltweit geht man immerhin von 625 Millionen Computer-Anwendern aus, von denen 95 % einen PC benutzen[2]. Ein gewisses Beharrungsvermögen bringt der Ersteller erst dann auf, wenn eine genügend grosse Benutzerzahl erreicht ist. Vorher nimmt er ggf. auch ganz Abstand von der Weiterentwicklung der Software. Nur selten wird der Ersteller aber professionelle Marktstudien o.ä. betreiben, da das in seiner Vorstellung den Aufwand nur künstlich in die Höhe treibt und er seine Kosten im Falle eines Fehlschlages gering halten möchte. Anspruchsvolle und innovative Software ist von diesem Typus des Softwareerstellers eher nicht zu erwarten.

n) weil man darüber Hardware und Services verkaufen möchte

Die Software wird als Enabler für den Verkauf von Hardware und Services benutzt. So fördert beispielsweise IBM die Entwicklung von Linux, während die Haupteinnahmen aus den Bereichen Hardware und Software stammen[3]. Man könnte einerseits die Hardware nicht verkaufen, sofern man nicht Software bereitstellt, um diese nutzen zu können[4].

1. Pro Minute Rechnerausfall muss man im Bankbereich, insbesondere im Börsenumfeld, bereits mit mehreren Millionen Schaden rechnen.
2. Sicherheitsreport 1/2002, S. 4, herausgegeben von der Verwaltungs-Berufsgenossenschaft, http://www.vbg.de.
3. Dort stammen etwa 80 % der Einnahmen aus den Bereichen Services (40 %) und Hardware (39 %), Software macht nur 15 % aus, während sie gerade einmal 4,2 % der Kosten verursacht, http://www.ibm.com/annualreport/2001/financial_reports/fr_cfs_earnings.html.
4. Betriebssystem, Compiler, Treiber.

Andererseits kann man mit Software auch Bedarf nach schnellerer Hardware erzeugen, indem man nützliche Funktionen anbietet, die aber grössere Rechenkapazitäten voraussetzen[1]. Bei den Services ist es traditionell so, dass primär die für ihr Knowhow bekannten Unternehmen profitieren. Das sind einmal die alteingesessenen Firmen mit jahrzehntelanger Erfahrung und eigener Forschung. Aber zunehmend kommen auch junge und kleine Unternehmen in Frage, sofern sie die Software selbst entwickelt haben. Das folgt aus der Überlegung, dass der Entwickler seine Software typischerweise am besten kennt. Ein Hinderungsgrund kann jedoch darin bestehen, dass sie zu geringe Ressourcen haben, um die Aufträge zu realisieren[2].

o) weil man so Marketingdaten erhält

Eine Reihe von Softwareprodukten werden nur entwickelt, um an die Adressdaten des Benutzers zu kommen. Dazu gehört auch sogenannte Spyware, die den Vorlieben des Benutzers auf die Schliche kommen will und die Ergebnisse dann (in der Regel ohne Wissen des Benutzers) an den Hersteller zurücksendet[3]. Als wertvolle Marketingdaten gelten einmal die Mailadresse des Benutzers. Hinzu kommen aber oft auch Daten über das Betriebssystem und die Rechnerausstattung. Bei „Musiktauschbörsen“ geht es vor allem darum, herauszubekommen, welcher Musikmix den Benutzer interessiert, um daraus einerseits Konsequenzen für das normale Musikgeschäft abzuleiten, andererseits aber dem Kunden auch zielgruppengerechte Werbung zukommen zu lassen, damit er Umsätze tätigt.

p) Zusammenfassung

Es gibt eine Vielzahl von wirtschaftlichen Gründen, aus denen Einzelpersonen oder Unternehmen heraus Software entwickeln.

2. aus rechtlichen Gründen

a) weil Verträge Wiederverwendung verbieten

Ein Phänomen ist die Doppelentwicklung von Software, obwohl eine ähnliche oder sogar die gleiche Software bereits im Hause existiert. Das geschieht in grösseren Softwarefirmen oft deshalb, weil man die Software im Kundenauftrag entwickelt hat und dem Kunden vertraglich zu-

1. Mehrdimensionales OLAP (Online Analytical Processing), Datamining.
2. Wenn z.B. ein Einzelentwickler eine Software erstellt, kann er kaum ernsthaft Service für die Nutzer anbieten, da er dann nicht mehr zum Weiterentwickeln kommt. Auch sind Zeiten von Freizeit oder Urlaub problematisch. Solche Aufträge kann er nicht realisieren ohne zu expandieren.
3. Vgl. die Beispiele in http://www.heise.de/newsticker/data/pab-09.01.02-000 oder http://www.heise.de/newsticker/data/lab-03.01.02-000.

gesichert hat, die von ihm finanzierten Arbeitsergebnisse nicht anderweitig zu nutzen[1]. Die Auftraggeber lassen sich derartige Bedingungen garantieren, weil sie sonst fürchten, die Softwarefirma könnte aus der einstigen Individualsoftware mit etwas Aufwand eine Standardsoftware machen und diese auch Mitbewerbern günstig anbieten. Das ist natürlich aus Wettbewerbsgesichtspunkten heraus nicht wünschenswert. Auch Anbieter von Softwarekomponenten verwenden solche Verträge, um zu verhindern, dass ein Lizenznehmer die Softwarekomponente geringfügig „verpackt“ und erfolgreicher als der Komponentenhersteller vermarktet[2].

b) weil Know-How ungenügend geschützt ist

Für den Schutz von Knowhow gibt es nur beschränkt wirksame Möglichkeiten[3]. Sofern die Idee nicht patentierbar ist, bleibt oft nur der vertragliche Schutz über strafbewehrte Knowhow-Verträge, um eine Verbreitung des Wissens zu verhindern. Wenn nun also ein Auftraggeber dem Softwarehersteller Einblick in sein Knowhow geben muss, riskiert er damit die Weitergabe an Dritte. Wenn es um sensible Bereiche geht, wird daher häufig Software lieber im eigenen Unternehmen entwickelt, als den Auftrag extern zu vergeben.

c) weil Patente verletzt wurden

Vielfach wird erwähnt, dass Software in den USA aufgrund der Patentlage neu entwickelt oder zumindest geändert werden muss[4]. Grundsätzlich ist Software als solche in Deutschland bislang nicht patentierbar[5], so dass derartige Probleme hier nicht bekannt sind. Jedoch wird seit geraumer Zeit, u.a. auf EU-Ebene[6], vorgeschlagen, eine Patentierung zu ermöglichen.

d) Zusammenfassung

Es gibt einige rechtliche Gründe für die Entwicklung von Software. Zum Schutz von Knowhow werden interne Eigenentwicklungen durchgeführt, die ökonomisch ineffizient sind, da Software redundant entwickelt wird. Ein grundsätzlicher Patentschutz für jedwede Software

1. Ähnlich: BMBF S. 85 zum Know-How-Schutz.
2. Das ist natürlich nur dann möglich, wenn der Lizenznehmer nicht pro Benutzung der Komponente bezahlen muss, sondern z.B. pro Entwicklerarbeitsplatz abgerechnet wird (typisch für Entwicklerbibliotheken, bei denen der Lizenzgeber einen fixen Lizenzpreis vorzieht, da er sich nicht am vermeintlichen Erfolg des Lizenznehmers beteiligen möchte).
3. BMBF S. 181; vgl. „e) ist durch UWG geschützt” auf Seite 17.
4. Vgl. ähnlich: BMBF S. 174.
5. Vgl. „d) ist in Spezialfällen durch Patentrecht geschützt” auf Seite 17.
6. Vgl. http://www.heise.de/newsticker/data/anw-20.02.02-005.

könnte einerseits zur ineffizienten Entwicklung von Umgehungslösungen führen. Andererseits könnten durch die vorhergehende Recherche ineffiziente Entwicklungen vermieden werden.

3. aus technischen Gründen

a) weil Sicherheitsbedenken bestehen

Eine bestimmte Klasse von Software ist so sicherheitskritisch, dass sie im eigenen Unternehmen überprüft werden muss. Bei lizensierter Software ist das häufig nicht möglich, weil der Hersteller sich weigert, den Quelltext zur Verfügung zu stellen, die entsprechende Quelltext-Lizenz zu teuer ist oder Lizenzbedingungen inakzeptabel sind. Es gibt auch den Fall, dass die Entwicklung selbst bereits sicherheitskritisch ist, z.B. weil dazu Geschäftsgeheimnisse verwendet werden müssen.

Aufsehen erregte die Beteiligung einer Scientology-Firma am Defragmentierungsprogramm von Microsoft Windows 2000[1]. Dabei stand zu befürchten, dass Dritte unbemerkten Zugriff auf sensible Daten auf der Festplatte erlangen. Schwerwiegender war die behauptete Kooperation von Microsoft mit dem US-Geheimdienst NSA[2]. Danach sollen Microsoft, Netscape und Lotus dem Geheimdienst Zugang zu Emails verschaffen. Dies sei im Rahmen von Wirtschaftsspionage zu sehen[3]. Ferner können unbemerkte Mikrofonaufzeichnungen gemacht werden[4]. Übliche Software warnt den Benutzer nicht einmal davor[5]. Die Verantwortung dafür liegt vor allem beim Betriebssystem[6]. Über den integrierten Webbrowser können ungefragt lokale Dateien ausgelesen und an Angreifer übermittelt werden[7]. Das Chat-Programm ermöglicht das Ausführen beliebiger Programme durch Angreifer auf dem Rechner des Nutzers[8]. Auch andere Programme sind äusserst anfällig[9].

b) damit die Datenstrukturen bekannt sind

Wenn Software lizensiert wird, so sind häufig die von der Software verwendeten Datenstrukturen nicht bekannt. Diese werden aber benötigt, um Daten über lange Zeiträume verfügbar zu halten[10]. Zwar könnte

1. Vgl. http://www.heise.de/ct/99/25/058.
2. Vgl. http://www.heise.de/newsticker/data/cp-20.02.00-000.
3. Vgl. http://www.heise.de/newsticker/data/ame-11.05.99-000 und http://www.spiegel.de/politik/ausland/0,1518,121349,00.html.
4. Vgl. http://www.heise.de/newsticker/data/wst-26.03.01-003 und http://www.spiegel.de/netzwelt/politik/0,1518,125037,00.html.
5. Vgl. http://www.spiegel.de/netzwelt/politik/0,1518,125030,00.html.
6. Weil dieses per Definition für die Kontrolle sämtlicher Resourcen zuständig ist.
7. Vgl. http://www.heise.de/ct/browsercheck/demos.shtml.
8. Vgl. statt vieler: http://www.heise.de/newsticker/data/dwi-09.05.02-001.
9. Vgl. die Übersichten bei http://www.securityfocus.com (Top Ten Attacks bzw. Vulnerabilities), http://www.cert.org oder http://www.bsi.de.

man versuchen eine entsprechende Dokumentation zu erwerben. Jedoch besteht ohne Quelltext das Risiko, dass die Dokumentation fehlerhaft ist. Eine Reihe von Softwareprodukten verwendet darüber hinaus (gemessen an den Anforderungen im Unternehmen) unnötig aufwendige Datenstrukturen[1]. Daher entschliessen sich Firmen dazu, Software selbst zu entwickeln.

c) damit Standards genutzt werden

Man stellt fest, dass Software entwickelt wird, obwohl bereits ausreichende Programme vorhanden sind. Man entwickelt die Software aber neu, weil man Standards nutzen möchte[2]. Davon erhofft man sich zukünftig Vorteile, wenn es um die Interoperabilität oder die Erweiterung mit Standardkomponenten geht.

d) um Wiederverwendung zu ermöglichen

Vereinzelt werden grosse Softwareprojekte gestartet, um eine einheitliche Plattform für Branchenstandards zu schaffen[3]. Derartige Frameworks sollen als Skelett einer Software zwecks Wiederverwendung[4] angeboten werden, wodurch nur noch die speziellen, anpassungsbedürftigen Teile hinzugefügt werden müssen. In diesem Zusammenhang wird Software gerne „auf Vorrat“ entwickelt. Das bedeutet, dass man zukünftig eventuell benötigte Funktionalität versucht zu erkennen und zumindest die Schnittstellen dafür zu schaffen. Mehrere derartige Projekte sind allerdings an der Komplexität gescheitert[5].

e) weil gute Werkzeuge fehlen

Man kann Softwareentwickler in einer Hinsicht gut mit Heimwerkern vergleichen: Wenn ihnen ein bestimmtes Werkzeug[6] fehlt und es sich nicht kaufen lässt[7], versuchen sie es selbst zu bauen. Das ist naheliegend, da das entsprechende Knowhow und „Material“[8] häufig vorhanden ist[9]. Die Möglichkeit der Eigenerstellung wird insbesondere dann

10. vgl. zum „Gedächtnisschwund der Informationsgesellschaft“ durch neue Datenformate: http://www.spiegel.de/spiegel/21jh/0,1518,82510,00.html.
1. So z.B. wenn man Microsoft Word benutzt, um einen einzigen Typ von Standardbrief zu schreiben. Die Analyse der erzeugten DOC-Datei erfordert auch die Auseinandersetzung mit den nicht verwendeten Fähigkeiten der Software.
2. BMBF S. 126.
3. Z.B. das Cheops-Projekt von RWE, eingestellt nach 4 Jahren Planungs-/Implementierungszeit und 100 Millionen DM Aufwendungen, vgl. http://www.computerwoche.de/index.cfm?pageid=267&type=ArtikelDetail&id=144001.
4. BMBF S. 125.
5. Z.B. Cheops (s.o.) oder Euro-Elan, Projektkosten rund 70 Millionen DM, vgl. http://www.computerwoche.de/index.cfm?pageid=267&type=ArtikelDetail&id=6245 oder http://www.berlinonline.de/wissen/berliner_zeitung/archiv/1995/0118/wirtschaft/0153/index.html.
6. BMBF S. 133.
7. Zu teuer ist, kein Anbieter bekannt ist usw.
8. Beispielsweise Compiler oder benötigte Hardware.

gerne genutzt, wenn die Anforderungen sehr speziell sind und man nicht davon ausgeht, dass sie in einem Fremdprodukt ansprechend gelöst werden.

f) zu Forschungszwecken

So wie ein Schriftsteller oder Handwerker üben muss, wenn er über neue Themen schreibt bzw. neue Werkzeuge einsetzt, so muss auch der Ersteller von Software üben. Darüber hinaus gilt es Forschung zu betreiben, um die Möglichkeiten und Grenzen neuer Werkzeuge und Methoden zu erkunden. Zu diesem Zwecke wird auch Software erstellt. Häufig werden im Vorfeld oder auch im Rahmen eines Projektes Forschungsarbeiten oder Vorstudien erstellt[1]. Diese haben oft einen selbständigen Charakter, sind also mit etwas Aufwand durchaus zu einem nutzbringenden Programm aufrüstbar. Wenn das Projekt abgebrochen oder abgeschlossen wurde, werden diese Arbeiten oft nicht mehr benötigt und veralten schnell.

g) wegen Scheitern des Vorgängerprojekts

Leider erfüllen nicht alle Softwareprodukte die qualitativen Anforderungen[2]. Gerade bei externer Auftragsvergabe und Werkverträgen stellt sich das oft erst heraus, wenn der Kunde die Abnahme verweigert. Eine bedeutende Anzahl von Softwareprojekten (53 %) gerät in Schwierigkeiten, eine beachtliche Zahl wird ganz abgebrochen (30 %)[3]. Dafür gibt es eine Vielzahl von Gründen[4]. In der Regel besteht der Bedarf an einer Problemlösung aber weiterhin. Daher werden neue Softwareprojekte initiiert, ggf. mit anderen Schwerpunkten und Eckdaten. Die Software wird so teilweise doppelt entwickelt.

h) weil sie veraltet ist

aa) Änderung der fachlichen Umgebung

Damit ist gemeint, dass sich die fachlichen Anforderungen an die Software durch Zeitablauf ändern. Beispielsweise betrifft das Änderungen in der Steuergesetzgebung, den Bilanzierungsrichtlinien, den Aufbewahrungsvorschriften für Akten, Änderungen am Geschäftsverteilungsplan, Hinzukommen oder Wegfallen von Zuständigkeiten oder

9. Es reicht vielleicht nicht für die beste Lösung, aber es wird für den Einzelfall funktionieren. Eine Regel der Informatik lautet: Provisorien halten am längsten.
1. BMBF S. 139, 145 f., 162.
2. BMBF S. 160.
3. Studie der Standish Group, http://www.standishgroup.com/chaos.html; die allerdings jede Abweichung von Termin/Kosten/Qualität bereits als Problem definierte.
4. Ausführlicher dazu: Zahrnt, S. 5 bis 12.

das Zusammengehen oder Abtrennen von Geschäftsbereichen. Derartige Änderungen sind sehr häufig. Man versucht zwar die Auswirkungen auf die Software so gering wie möglich zu halten[1]. Es lässt sich nicht sagen, ob Entscheidungen zugunsten von Neuentwicklungen oder zur Pflege der Software überwiegen. Je grösser das System ist, desto häufiger wird wohl die Pflege den Vorzug erhalten[2].

bb) Änderung der technischen Umgebung

Nur in wenigen Bereichen gelingt es, die Hardware-Umgebung über längere Zeit konstant zu halten. Zu nennen ist hier vorrangig der Grossrechner- und z.T. der Unix-Bereich, wo die Änderungsrate bei 10 Jahren oder darüber liegen dürfte. Die typische Grossrechnerumgebung MVS, Cobol, CICS und Fortran ist konzeptionell etwa 40 Jahre alt, die Unix-Systeme[3] sind mit 20-30 Jahren etwas jünger. Hier herrscht also eine hohe Stabilität. Im PC-Bereich hingegen kann man davon ausgehen, dass sich die Umgebung im Abstand von 3 Jahren[4] gravierend ändert und somit bedeutsame Änderungen auch auf die Software durchschlagen.

cc) weil alte Software erweitert oder ausgetauscht wird

In grossen Unternehmen ist in der Regel so genannte Legacy-Software vorhanden. Vereinzelt wird diese durch neue Systeme erweitert oder ausgetauscht. Als Beispiel hierfür kann Banksoftware dienen, die mit einer Schnittstelle zur Kontostandabfrage über das Internet versehen wird. Im Rahmen der Jahr-2000- oder Euro-Umstellung wurden ebenfalls eine Reihe von Systemen ausgetauscht. Aber auch Konsolidierung kann ein Argument für die Entwicklung neuer Software sein[5].

dd) weil alte Software gepflegt/reengineered wird

Keine Software ist so gut, dass sie nicht noch verbessert werden könnte[6]. In diesem Bewusstsein wird Software in vielen Zyklen[7] überarbeitet und verbessert[8]. Es ist eher die Regel als die Ausnahme, dass für jede

1. Das war eines der Versprechen objektorientierter Softwareentwicklung. Mittlerweile ist diesbezüglich Ernüchterung eingekehrt, weil man feststellt, dass es nach wie vor eher eine Frage der Architektur und geeigneter Modularisierung ist.
2. Beispielsweise bei Banken und Versicherungen.
3. Darunter sollen hier professionelle Workstations wie von Sun Microsystems oder die AIX-Rechner von IBM verstanden werden, nicht PC-Unixe.
4. Das entspricht auch der üblichen Abschreibungsdauer für PCs und Software, http://www.bundesfinanzministerium.de/BMF-.336.5067/.htm.
5. Angeblich hat allein IBM etwa 36 verschiedene Betriebssysteme für ihre Hardware im Programm, die man mit Linux z.T. ablösen kann, http://www.heise.de/newsticker/foren/go.shtml?read=1&msg_id=1708910&forum_id=28956.
6. Vgl. Eric S. Raymond, S. 25. Er empfiehlt, die erste Version einer Software wegzuwerfen und sie grundlegend neu zu erstellen, weil man bei der ersten Erstellung ungemein viel über die Aufgabe gelernt hat und sich dieses Wissen für eine fundamental verbesserte zweite Version nutzen lässt.

Zeile Quelltext, die in der Endversion enthalten ist, unzählige Zeilen geschrieben worden sind, die verworfen wurden.

Diese Zwischen- oder Vorgängerversionen sind häufig durchaus nutzbar. In der Regel werden sie aber nicht vermarktet. Einige Hersteller senken den Preis des Vorgängers bei Erscheinen der Nachfolgeversion. Sie erhoffen sich dadurch eine erhöhte Marktdurchdringung, weil ihre Vorgänger-Produkte – ohne in Erklärungsnot zu geraten[1] – günstiger sein können als die (oft recht teuren) aktuellen Versionen und damit den Mitbewerbern Marktanteile abnehmen, gleichzeitig aber einen Updatepfad[2] auf die aktuelle Version eröffnen.

ee) Zusammenfassung

Auf den ersten Blick scheint Software als digitales Gut nicht „schlecht werden“ zu können[3]. Tatsächlich ist die durchschnittliche Einsatzdauer von Software jedoch sehr gering, wenn man den Zeitraum betrachtet, während dessen sie nicht in grösserem Umfang gepflegt werden muss. Diese kann man auf etwa 18 Monate abschätzen[4]. Das liegt an der sich häufig ändernden Umgebung[5] ebenso wie an neuen Anforderungen[6].

i) Zusammenfassung

Es gibt eine Vielzahl von technischen Gründen für die Entwicklung von Software. Wichtige Aspekte sind Sicherheit, Standards, Forschungszwecke und die Überarbeitung bzw. der Austausch von Software.

4. ohne eine formale Qualifikation dafür zu haben

Der Zugang zur Softwareentwicklung ist nicht durch Gesetze reguliert.

---

7. Vgl. das RUP-Prozessmodell, welches ein iteratives, inkrementelles Vorgehen fordert, bei dem man sich in vielen kleinen Zyklen spiralförmig dem Ziel nähert. Es hat sich gezeigt, dass dieses Vorgehen Vorteile bietet, weil in den Projekten anfangs nicht klar ist, wie genau das Ziel aussieht. Daher ist es einfacher und risikoärmer, sich schrittweise vorzutasten, statt auf den grossen Wurf zu hoffen, mit dem man das Ziel auf direktem Wege erreichen könnte.
8. Redesign/Reengineering.
1. Kunden reagieren oft irritiert, wenn ein gerade erworbenes Produkt plötzlich drastisch günstiger zu haben ist.
2. Dieser ist nicht ganz unwichtig. Einerseits möchte man den Mitbewerbern Kunden ausspannen. Andererseits muss man dies in recht engen Grenzen eines fairen Wettbewerbs vollziehen, was bei Dumpingpreisen nur für Kunden anderer Unternehmen nicht unproblematisch ist. Das neutrale Anbieten alter Versionen zu reduziertem Preis ist hingegen aus Wettbewerbssicht problemlos möglich. Da der Kunde sich an die Software gewöhnt, wird er bei Gefallen auf die aktuelle Version umsteigen, sofern ihn der hohe Preis nicht abschreckt. Kunden, die den Preis der aktuellen Version nicht zahlen können oder wollen, sind für den Anbieter jedoch eh verloren. Mit Hilfe der günstigen alten Versionen kann der Anbieter noch Geld abschöpfen. Dafür muss er im Gegensatz zur Entwicklung von reduzierten Versionen keinen zusätzlichen Aufwand treiben.
3. Vgl. zur Alterung: Balzert S. 26.
4. Vgl. BMWi S. 63.
5. Damit ist die Hardware, das Betriebssystem und die verwendeten Programmierwerkzeuge (Compiler, Bibliotheken, Tools) gemeint.
6. Bei einer Steuerberechnungssoftware sind z.B. neue rechtliche Regelungen zu berücksichtigen.

Jeder, der sich dazu befähigt fühlt, kann Software entwickeln und vertreiben[1]. Da es ja auf den ersten Blick nur darum geht, ein paar Programmzeilen zu schreiben und so lange zu probieren, bis es klappt, ist die Einstiegshürde gering[2]. In der Praxis wird das Erfordernis formaler Qualifikationen für Standardsoftware sogar weithin abgelehnt. Softwareentwicklung ist einmal künstlerische Tätigkeit[3], andererseits aber auch dem Maschinenbau ähnlich[4]. Durch mangelhafte Software entstehen für den Entwickler nur selten gefährliche Ansprüche Dritter[5]. Daher ist die Einsicht nicht verbreitet, dass es vielleicht besser wäre, eine formale Qualifikation zu erlangen. Den Kunden ist meist egal, wer die Software entwickelt hat. Das ist ein Unterschied zu anderen Branchen. Kaum jemand würde wohl akzeptieren, dass ein Historiker die Bremsen des Autos entwickelt hat oder ein Informatiker die Gehirnoperation durchführt, oder wäre mit der Erklärung zufrieden, es handele sich um ein Wunderkind oder Naturtalent. Problematisch ist, dass Software bei einigen Testfällen funktionieren kann, jedoch bei anderen, geringfügig abweichenden Werten abrupt versagt. Ein vollständiger Test ist in der Regel aus Komplexitätsgründen nicht möglich[6]. Daher lässt sich die Qualifikation des Erstellers nicht allein durch Testergebnisse ersetzen[7]. Zwar haben 63 % der angestellten Softwareentwickler eine akademische Ausbildung[8]. Jedoch haben nur 47 % Informatik studiert[9], das sind bezogen auf die Gesamtzahl der Entwickler weniger als 30 %. Hinzu kommt, dass die universitäre Informatik ein weites Feld ist, auf dem

1. Das ist auch üblich: Schüler, Studenten informatikfremder Fachrichtungen und Nichtinformatiker als Arbeitnehmer entwickeln Software. Oft genügen wenige kurze Tests und die Software wird veröffentlicht.
2. Die Ergebnisse können im Rahmen von Reviews blankes Entsetzen bei Informatikern hervorrufen. Der Quelltext ist oftmals eine reine Sammlung von Anti-Pattern (vgl. das gleichnamige Buch „AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis“ von William J. Brown, 1998) und reagiert mehr zufällig auf Eingaben denn systematisch, weitab jeder Ingenieurtradition.
3. Der Ersteller einer Software gleicht einem Schriftsteller, der ein Buch schreibt, weil auch Sourcecodes längere Texte sind. Letzerer benötigt keine formale Qualifikation, weil man der Meinung ist, er könne mit einem schlechten Buch nicht viel Schaden für die Gesellschaft anrichten.
4. Der Computer als Maschine ändert durch die Software sein Verhalten. Ein Softwareentwickler baut somit ständig neue Maschinen. Allerdings kann er die Möglichkeiten der vorhandenen Maschine nur nutzen, aber nicht erweitern.
5. Gewährleistung, Schadenersatz, Produkthaftung usw. werden bei Standardsoftware eher selten geltend gemacht. Bei Individualsoftware ist das anders.
6. Vollständig bedeutet in diesem Zusammenhang, dass sämtliche Pfade durch das Programm mit allen möglichen bzw. relevanten Eingabewerten ausgetestet werden müssten. Aus kombinatorischen Gründen ergeben sich selbst bei eher kleinen Programmen schon Millionen von Kombinationen. Selbst mit automatisierten Tests sind diese oft nicht durchführbar.
7. Der Ersteller muss bei seinem Vorgehen bemerken, welche Situationen kritisch werden könnten. „Beliebte“ Fehler sind Race Conditions und Deadlocks, vgl. http://www.boost.org/libs/thread/doc/definitions.html, die sporadisch auftreten und nur selten vom Kunden reproduziert werden können.
8. BMBF S. 101.

Software Engineering nur ein Gebiet von vielen darstellt[1].

5. Zusammenfassung

Software wird aus wirtschaftlichen, rechtlichen und technischen Gründen neu entwickelt, mehrheitlich durch Nicht-Informatiker.

A 4. Urheber entwickeln Software nicht

1. weil Patentverletzungen befürchtet werden

Gerade kleinere Softwarehersteller werden sich eher völlig aus der Softwareentwicklung zurückziehen, als sich einem langwierigen und teuren Patentstreit zu stellen[2]. Die Unternehmen des Primärsektors sind überwiegend sehr klein[3]. Es kommt hier kaum darauf an, ob tatsächlich eine Patentverletzung vorliegt oder man z.B. die Löschung beantragen könnte. Der psychologische Effekt (Einschüchterung) ist viel bedeutender, besonders wenn der Patentinhaber übermächtig erscheint.

2. weil sie die Entwicklungskosten nicht vorstrecken können

Es ist selten, dass wenig aufwendige Entwicklungen zu einer stabilen Einnahmeplattform führen. Die Entwicklung wirtschaftlich erfolgreicher Software kostet in der Regel sehr viel Geld[4]. Erfolgreiche Produkte setzen auch im Softwaremarkt entweder eine Marktnische oder solides Expertenwissen in der Anwendungsdomäne voraus.

a) Marktnischen

Marktnischen sind im Softwaremarkt gar nicht so selten. Das liegt daran, dass ständig neue Standards entstehen und es naturgemäss noch keine Produkte geben kann, die diese Standards bedienen[5]. In diesem Fall muss sich ein Softwarehersteller also „nur“ ansehen, welche neuen Standards zu erwarten sind[6] und dazu ein Produkt entwickeln. Mit zu-

9. BMBF S. 103; es folgen (in abnehmender Häufigkeit) Elektrotechnik, BWL/ VWL, Physik, Mathematik, Wirtschaftsinformatik, Maschinenbau, Nachrichtentechnik, Chemie. Immerhin 19 % stammen aus sonstigen Studiengängen.
1. Andere Bereiche sind z.B. Rechnerarchitektur (hardwarelastig, sehr nahe an Elektrotechnik) und theoretische Informatik (sehr nahe an der Mathematik). Diese haben zusammen im Studium selbst bei entsprechender Schwerpunktsetzung meist einen höheren Umfang als das Software Engineering. Das allgemeine Informatikstudium ist aufgrund seiner Geschichte sehr mathematiklastig. Erst in den letzten Jahren lässt man mehr Anwendungsorientierung zu, u.a. in speziellen Studiengängen „Angewandte Informatik“ oder „Software Engineering“. Das führt dazu, dass man Informatikern gerne nachsagt, von allem eine Ahnung zu haben, aber paradoxerweise kaum etwas über Software-Entwicklung zu wissen.
2. Vgl. ähnlich: BMBF S. 174.
3. 77% der Unternehmen haben nur 1-9 Mitarbeiter, vgl. BMBF S. 58.
4. Als Mindestaufwand bei nicht-trivialer Software wird man von mehreren 100.000 Euro ausgehen können, die vor einem Break-Even zu investieren sind. Das ist aber eine im Vergleich zu anderen Branchen geringe Hürde, ggf. besteht ein Grossteil nur aus Opportunitäts- und nicht realen Kosten.
5. So beispielsweise die relativ neuen XML-Standards unter http://www.w3.org.
6. Am besten durch Mitarbeit in einem Standardisierungsgremium oder Verfolgen der Fachpublikationen (z.B. http://www.acm.org, http://www.ieee.org oder http://www.ni.din.de).

nehmender Komplexität der Standards ist deren Implementierung allerdings gar nicht so einfach. Dennoch: Wenn es ein Hersteller schafft, ist er einer von wenigen[1]. Die Implementierung ist aber eine Kostenfrage. Bei einem typischen, nicht-trivialen W3-Standard wird man für einen Informatiker mindestens ein Mannjahr bis zum Produkt voraussetzen können. Diese Zeit muss vorfinanziert werden. Der Schwerpunkt dürfte auf der Arbeitsleistung liegen[2], da Rechner und Software bei Softwareherstellern sowieso im wesentlichen vorhanden sind[3]. Bei hardwarenahen Entwicklungen ist das anders[4]. Hier müssen ggf. Entwicklerversionen der Hardware beschafft werden.

b) Expertenwissen in der Anwendungsdomäne

Ein Softwarehersteller kann sich auch eine Anwendungsdomäne[5] suchen und dafür eine Software entwickeln. Software bildet jedoch zu einem grossen Teil lediglich das Wissen ihrer Autoren ab. Daher ist ein Produkt nahezu unverkäuflich, wenn der Hersteller keinen Zugang zu Experten dieser Anwendungsdomäne hatte. Grundsätzlich gibt es zwei Möglichkeiten für den Hersteller. Er kann Experten auf dem Markt einkaufen oder selbst mit eigenen Resourcen zum Experten werden. Der Einkauf von Experten ist riskant. Einerseits sind echte Experten nicht leicht verfügbar[6] und müssen vorfinanziert werden. Andererseits ist der Zugriff auf einen Experten noch keine Gewähr dafür, dass die Softwareentwicklung gelingt[7]. Durch die Ausbildung eigener Resourcen zu Experten werden zwar Loyalitätsprobleme vermieden. Andererseits dürfte es in den seltensten Fällen gelingen, an die Qualität wirklicher Experten auch nur heranzukommen. Die Schwierigkeit besteht darin, einerseits das technische Knowhow, andererseits aber auch das Wissen der Fach-

1. Vgl. zum Marketing in extremen Marktnischen, z.B. mit nur 1000 potentiellen Anwendern weltweit: Moore S. 77 f.
2. Sofern man diese mit Opportunitätskosten bewertet – also realen Marktpreisen für Informatik-Dienstleistungen – und nicht nur von der selbstlosen Sicherung des Existenzminimums ausgeht.
3. Ganz selten benötigen Softwareentwicklungen für den Massenmarkt spezielle und teure Hardware. Im allgemeinen wird für die PC-Plattform entwickelt, wofür die Entwicklungswerkzeuge weitestgehend vorhanden sind.
4. Beispielsweise bei der Entwicklung von Software für Linux-Handhelds.
5. Z.B. Rechnungswesen oder Lernsoftware.
6. Echte Experten sind oft bereits bei den Mitbewerbern angestellt oder haben kein besonderes Interesse, einem Softwarehersteller ihr Wissen zu verraten. Daher muss ihre Motivation durch entsprechende Vergütung gefördert werden.
7. Es mag dem Hersteller nicht gelingen, das Wissen des Experten in Software umzusetzen. Das ist besonders der Fall, wenn der Experte bislang nie in einem Softwareprojekt gearbeitet hat und es nicht versteht, sein Wissen „informatikergerecht“ weiterzugeben. Typisch dafür sind Strukturen, die Informatikern willkürlich oder zufällig erscheinen, weil der Experte wichtige Entscheidungskriterien nicht erwähnt, da sie für ihn zwar an Beispielen leicht beschreibbar, aber eben nicht berechenbar sind. Man stelle sich einen Juristen vor, der einem Informatiker beschreibt, wie man „Vorsatz“ automatisch von „Fahrlässigkeit“ trennen kann.

domäne zu haben. Man kann feststellen, dass viele fachlich ausgezeichnete Programme technisch weit hinter dem Stand der Technik zurückliegen[1]. Andererseits gibt es eine Reihe technisch ausgezeichneter Programme, die fachlich kaum nutzbar oder zumindest suboptimal sind[2].
Sofern der Softwarehersteller in einer für ihn neuen Domäne arbeiten möchte, kann sich nach umfangreichen Vorarbeiten herausstellen, dass eine Software für den angestrebten Einsatz prinzipiell nicht erstellbar ist[3]. Das Risiko ist dabei also sehr hoch. Gleichwohl erhöhen sich die Ertragschancen für denjenigen, der das Problem dennoch löst.
Sofern der Softwarehersteller das benötigte fachliche Knowhow nicht hat oder mit vertretbarem Aufwand beschaffen kann, wird er also von der Produktentwicklung eher Abstand nehmen.

c) Zusammenfassung

Mit Trivialprogrammen lässt sich nur selten Gewinn machen. Wenn man eine Marktnische entdeckt hat, müssen hohe Entwicklungskosten vorfinanziert werden. Problematisch kann auch die Verfügbarkeit von Expertenwissen sein, das man fachlich für die Software benötigt. Beide Punkte können dazu führen, dass ein Programm nicht entwickelt wird.

3. weil das Risiko zu hoch ist

Es ist schwer abzusehen, ob eine Software zum Break-Even oder gar Gewinnen führt. Selbst gute Softwareprodukte können aus Marketinggründen schwer absetzbar sein. Das mag daran liegen, dass die Zielgruppe schwer zu erreichen ist[4]. Oder sie weiss gar nicht, dass eine Software existiert, die ihre Probleme lösen kann[5]. Hinzu kommt die

1. Häufig: Nutzung von Microsoft Access als „Datenbankmanagementsystem“, Entwicklung mit fehlerträchtigen Sprachen wie Visual Basic, Fehlverhalten im Mehrbenutzerbetrieb, Race Conditions, Deadlocks usw.
2. Beispiel: Die Suchmaschine von Beck Online, http://www.beck.de. Sie findet beim Suchkriterium „Anfechtung“ tatsächlich nur dieses Wort, nicht aber damit zusammenhängende Formen wie „anfechten“, „angefochten“, „anzufechten“, was die Arbeit bei einer Volltextsuche in Urteilen letztlich auf den Benutzer abwälzt, ganz abgesehen von zahlreichen Verweisen auf Dokumente, die nicht einmal in eingescannter Form im System vorhanden sind (z.B. NJW vor 1980). Erst recht benutzen die Suchmaschinen keine echten oder umgangssprachlichen Synonyme (z.B. „Minderjähriger“ statt „beschränkt Geschäftsfähiger“).
3. Für einen juristisch Unkundigen wäre es sicher eine gute Idee, die Funktion des Richters oder auch Verwaltungsentscheidungen durch eine Software („Subsumtionsautomat“) zu ersetzen, damit die Verfahrensdauer verkürzt und Kosten reduziert werden. Auf den ersten Blick erscheint dieses Vorhaben sogar sehr einfach, da schliesslich „nur“ nach festgelegten Normen zu entscheiden ist.
4. Wenn sie nicht mit ausreichender Erfolgsquote z.B. über Fachzeitschriften oder Vereine/Verbände erreichbar ist. Vgl. EStamp, die mit ihren „Briefmarken per Internet“, Zielgruppe Freiberufler, Vereine und kleine Unternehmen, etwa $ 600-700 für die Anwerbung eines Kunden ausgeben mussten, siehe http://www.spiegel.de/wirtschaft/0,1518,192973,00.html und http://www.manager-magazin.de/netmanager/0,2828,154838-2,00.html.
5. Das ist bei Anwendungsbereichen der Fall, die bislang nicht oder nur wenig automatisiert sind.

Verantwortung, die man mit der Veröffentlichung eines Produktes auf sich nimmt. Es gibt beispielsweise Risiken bezüglich der Haftung für Schäden[1], sowie bei Patent-, Urheberrechts- und Markenverletzungen.

4. weil sie „Raubkopien“ fürchten

Theoretisch reicht eine Version des Programmes, die der Softwarehersteller aus der Hand gegeben hat, bereits aus, um diese Software unter Umgehung der Rechte des Herstellers weltweit zu verbreiten. Wenn der zweite Teil des Kinofilms „Der Herr der Ringe“ bereits im April 2002 in einer Rohfassung kostenlos im Internet verfügbar ist, lange bevor er im Dezember in die Kinos kommen soll, oder das Album der Musikgruppe Oasis drei Monate vor geplanter Veröffentlichung[2], dann ist dies auch bei Software zu befürchten. Man kann allerdings annehmen, dass Software mit einer grossen Zielgruppe sich zwar schnell auf „Tauschbörsen“ verbreitet, aber (absolut gesehen) auch viele Lizenznehmer findet. Software mit kleiner Zielgruppe ist in „Tauschbörsen“ weniger zu finden, so dass zumindest die relative Anzahl der Lizenznehmer gleich bleiben dürfte. Diese illegalen Verbreitungsformen stellen einerseits eine Gefahr dar. Andererseits tragen sie auch zum Bekanntwerden einer Software bei. Oft wird das Argument vorgetragen, die Hersteller würden diese Art der Verbreitung sogar billigen, wenn nicht sogar selbst forcieren, um ihr Produkt in den Markt zu bekommen. Nach dieser Argumentation lernen die Heimanwender das Programm kennen und bringen es später beim Arbeitgeber (hoffentlich lizensiert) zur Anwendung, so dass der Hersteller letztlich davon profitiert[3]. Letzlich wird der Hersteller abwägen, ob er die Investitionen angesichts des unsicheren Vertriebs in Kauf nehmen will.

5. weil Schnittstellen zu kompliziert sind

Die Entwickler distanzieren sich von der Erstellung eines Programmes auch, wenn die benötigten Schnittstellen zu kompliziert sind. Ein Unterfall dessen ist, dass die Schnittstellen nicht dokumentiert sind und erst selbst erforscht werden müssen (Reverse Engineering). Beispielsweise ist das Dateiformat von Microsoft Word dafür bekannt, schwer verständlich und nicht öffentlich dokumentiert zu sein[4]. Die Schnittstellen können Programm-Module betreffen oder auch Dateiformate. In der

1. Man nehme den neuen § 284 BGB (Ersatz vergeblicher Aufwendungen), der für den Softwarehersteller kaum kalkulierbare Risiken birgt.
2. Vgl. http://www.spiegel.de/kultur/musik/0,1518,192938,00.html.
3. Viele behaupten, z.B. die Microsoft-Produkte hätten sich nie durchgesetzt, wenn sie nicht in grossen Stückzahlen illegal Verbreitung gefunden hätten.

Regel sind Schnittstellen nicht standardisiert, sondern jeder Hersteller favorisiert seine eigenen Strukturen. Wenn es Standards gibt, so sind diese oft sehr lang[1], missverständlich oder auslegungsbedürftig, so dass sich im Detail wieder Unterschiede (und damit Fehlerquellen) ergeben. Beliebt ist auch die Erweiterung von Standards. Dabei wird ein Standard verwendet, aber mit zusätzlichen proprietären Festlegungen erweitert. Da man den Benutzer auf diese Erweiterungen nicht hinweist, verwässert dies den Standard im Laufe der Zeit bis zur Unbrauchbarkeit[2].

6. weil Softwarevertrieb aufwendig ist

Der herkömmliche Vertrieb von Software erfordert zusätzlich zur Herstellung eines Datenträgers (heute typischerweise CD oder DVD) den Druck von Handbüchern und einer Verkaufsverpackung. Softwareentwickler können auf spezialisierte Verlage zurückgreifen[3]. Dafür ist es jedoch einerseits erforderlich, dass die Software den Verlag interessiert, er ihr also ein gewisses Umsatzpotential zutraut. Das dürfte am ehesten bei Software der Fall sein, die einen grossen Kreis potentieller Nutzer hat, für die der Markt gleichzeitig aber noch nicht gesättigt ist[4]. Andererseits trifft das auf die wenigsten Softwareprodukte zu.

7. weil der Erfolg von Software nicht absehbar ist

Für den Entwickler eines Softwareproduktes ist schwer absehbar, inwiefern das Produkt am Markt erfolgreich sein wird. Das hängt damit zusammen, dass in der – oft recht langen – Zeit von den ersten Arbeiten am Produkt bis zur Fertigstellung andere Hersteller auf den Markt kommen können. In den umsatzversprechenden Hauptbereichen des Soft-

4. Man schreibe „Hallo“ in ein Word-Dokument und sehe sich mit einem Datei-/Hexadezimaleditor die aus diesen 5 Buchstaben bzw. Bytes generierte Doc-Datei in Binärform an. Sie ist bei Word 2000 immerhin 19.456 Bytes gross, enthält den eingegebenen Text gleich an 4 Stellen und ist alles andere als selbsterklärend. Gerüchteweise gibt es eine Dokumentation von Microsoft dazu, allerdings nur gegen Unterzeichnung eines strafbewehrten Non-Disclosure-Agreements.

1. Der W3-Standard HTML 4.01 umfasst etwa 250 Seiten, http://www.w3.org/TR/html401/html40.txt; SVG 1.0 hat 617 Seiten und referenziert zahlreiche andere Standards, http://www.w3.org/TR/2001/REC-SVG-20010904/REC-SVG-20010904.pdf.
2. Das beste Beispiel dafür ist HTML. Microsoft und Netscape haben ihm jahrelang eigene Anweisungen hinzugefügt, so dass die Kennzeichnung „Best viewed with Internet Explorer“ bzw. „… Netscape“ nötig wurde, weil Seiten, die für den einen HTML-Browser erstellt wurden, auf dem anderen nicht identisch angezeigt werden konnten. Auch die Datenbankabfragesprache SQL ist durch verschiedene Stufen gekennzeichnet, von denen namhafte Datenbankprodukte zwecks Behinderung eines Produktwechsels nur den kleinsten gemeinsamen Nenner unterstützen. Dieser macht die Abfragen jedoch deutlich langsamer, weil er bestimmte Join-Formen nicht unterstützt.
3. Beispielsweise Data Becker, http://www.databecker.de.
4. So gelten Textverarbeitungsprogramme trotz eines grossen Nutzerkreises als unverkäuflich, da Microsoft Word und StarOffice den Markt weitgehend unter sich aufteilen und jede Software an diesen starken Wettbewerbern gemessen wird. Eine Entwicklung mit wenigen Mannjahren hat demzufolge kaum eine Chance, über herkömmliche Wege erfolgreich vermarktet zu werden.

waremarktes kann man davon ausgehen, dass die Grosskonzerne[1] längst an entsprechender Software arbeiten. Marktstudien wären ein Mittel, solche Überraschungen zu vermeiden. Im sich schnell verändernden Softwaremarkt sind derartige Studien allerdings schwer durchzuführen und können nur die Produkte ergeben, die offensichtlich bereits am Markt vorhanden sind. Geheimgehaltene Neuentwicklungen werden jedoch von den grossen Herstellern oft mit einem enormen Resourceneinsatz in kurzer Zeit produziert, so dass sie in solchen Studien nie erscheinen können. Diese Produkte diktieren kleineren Herstellern aber die Preise, denn der Preis des „kleineren" Produktes kann nicht höher sein als der des Referenzproduktes.

8. weil Softwaremarketing schwierig und teuer ist

Um eine neue Software bekannt zu machen, ist ein hoher Aufwand erforderlich. Sofern der Softwarehersteller wenig Geld und wenig Erfahrung im Softwaremarketing hat, wird es für ihn schwierig, eine ausreichende Benutzermenge zu erreichen. Das liegt auch daran, dass Software ein komplexes technisches Produkt ist, und somit in höchstem Masse erklärungsbedürftig. Marketing für erklärungsbedürftige Produkte ist schwieriger und damit teurer. Typische Werbemöglichkeiten sind Mailings, Zeitschriftenanzeigen und Messestände.

a) Mailings

Grundlage eines jeden Mailings sind die Adressen. Sofern sie dem Werbenden nicht bekannt oder für ihn leicht selbst ermittelbar sind, muss er auf Adressdatenbanken zurückgreifen. Je nach Qualität[2] kostet eine Adresse dann von wenigen Cent bis zu mehreren Euro[3]. Dazu kommen die Kosten für Entwurf, Druck und Versand. Dennoch liegt die Rückläuferquote meist im Promillebereich, da sich die Empfänger nicht für das Produkt interessieren[4], die Werbung sie gar nicht erreicht hat[5] oder sie grundsätzlich nicht auf Werbung reagieren[6].

1. Beispielsweise Microsoft, IBM, Sun Microsystems, Oracle, SAP, Adobe.
2. Je spezieller die Selektion, desto teurer die Adresse, z.B. „Zahnärzte mit Praxis in guten Stadtteilen, die mehr als 3 Mitarbeiter haben, seit mindestens 5 Jahren am Markt sind und nur Privatpatienten behandeln".
3. Erfahrungswerte, genaue Zahlen erhält man z.B. über http://www.schober.de oder http://www.creditreform.de.
4. Allgemein keinen Bedarf haben, ein anderes Produkt bereits einsetzen oder gar nicht zur Zielgruppe gehören.
5. Mitarbeiter haben das Mailing entgegen genommen und aussortiert. So erreicht der Brief typischerweise die Sekretärin oder den Geschäftsführer, aber nicht den für die Softwarebeschaffung verantwortlichen Mitarbeiter.
6. Weil sie täglich zu viel nutzlose Werbung erhalten.

b) Zeitschriftenanzeigen

Hier kommen je nach Zielgruppe allgemeine Publikationen oder Fachzeitschriften in Frage. Die Zeitschriften haben allgemein recht hohe Anzeigenpreise[1]. Hinzu kommt, dass die Anzeigenabteilungen darauf hinweisen und es sich auch mit Marketingerfahrungen deckt, dass eine einmalige Schaltung nur einen geringen Effekt hat. Man wird hier eine Wiederholung im Monatszyklus einkalkulieren müssen. Problematisch kann die Selektivität werden, also das Verhältnis von potentiell interessierten Lesern zu nicht interessierten, besonders wenn für die Zielgruppe keine eigene Zeitschrift zur Verfügung steht[2].

c) Messestände

Sofern die Software für Fachbesucher von Messen interessant sein könnte, bietet sich ein Messestand an. Dieser sollte aber von Mailings und Zeitschriftenanzeigen flankiert werden, damit die Interessenten auch erfahren, dass es den Stand gibt. Damit steigt der Aufwand. Zusätzlich zu den Standgebühren[3] muss man noch die Standbauten und das Personal in Ansatz bringen. Dadurch sind kleinere bzw. junge Softwareunternehmen auf Messen kaum vertreten[4].

d) Redaktionelle Beiträge

Darüber hinaus kann man Pressemitteilungen an Fachzeitschriften senden und um die Rezension bitten. Das ist die billigste Version des Marketings, mit der sich potentiell die höchste Verbreitung erzielen lässt. Eine redaktionelle Erwähnung hat einen höheren Wert als jede Anzeige, da selbst eine schlechte Kritik noch den Eindruck vermittelt, die Redaktion habe das Produkt immerhin für im Markt bedeutsam gehalten.

e) Telefon-, Fax-, Mailmarketing

Wegen der Störung der Angerufenen und damit dem negativ besetzten Erstkontakt ist das Telefon-, Fax- oder Mailmarketing nur in Spezialbereichen sinnvoll, wenn sich wenige Personen für ein besonderes Pro-

1. Beispielpreise für eine ganze Seite vierfarbig im Redaktionsteil mit Angabe der verkauften Auflage: c't Magazin für Computertechnik, 383.530 Stück, 11.200 Euro, http://www.heise.de/mediadaten/print/ct/formate.shtml; Computerwoche, 42.444 Stück, 22.500 Euro, http://www.computerwoche.de/index.cfm?pageid=263&artid=27769&type=detail&category=54; Computer Bild, 990.680 Stück, 21.528 Euro, http://www.computerbild.de/media/CoBi2002.pdf; Der Spiegel, 1.065.625 Stück, 46.000 Euro, http://media.spiegel.de/objektinfo/o_spiegel/spiegel_frame_tarif.html; oder Heise Online, 41 Euro pro 1000 Bannereinblendungen, etwa 10.653.547 Einblendungen pro Monat für die Hauptseite, http://www.heise.de/mediadaten/online/preise.shtml.
2. Das ist häufig der Fall, da sich hoch spezialisierte Zeitschriften nur schwer am Markt behaupten können.
3. 180 Euro Grundmiete plus 46 Euro Servicegebühren pro Quadratmeter auf der CeBIT, ca. 700.000 Besucher, http://www.cebit.de/21839.
4. Oder nur über Partnerstände, d.h. Teilbereiche auf dem Stand grosser Firmen.

dukt interessieren könnten, die eine hohe Multiplikatorwirkung oder Entscheidungskompetenz haben[1].

f) Verschenken von Einstiegsversionen

Das ist die mit Abstand erfolgreichste Spielart des Softwaremarketings. Sie wird extensiv bei Shareware eingesetzt. Die Grundidee ist, dass alle herkömmlichen Marketingmöglichkeiten zu teuer und riskant sind sowie die Zielgruppe überwiegend verfehlen. Man erlaubt den Herausgebern von Shareware-CDs oder Zeitschriften, das Produkt auf CDs zu pressen, überspielt es auf einschlägige Internet-Sites und wartet darauf, dass der Nutzer nach dem Programm sucht oder es ausprobiert. Gute Software werde sich dann von allein am Markt durchsetzen. Insbesondere hofft man auf den Effekt von Empfehlungen zufriedener Nutzer im Bekanntenkreis. Das Geld für Marketing steckt man lieber in die Weiterentwicklung des Produktes und hält so den Preis niedrig[2].

g) Zusammenfassung

Das Marketing für Software kann über Mailings, Anzeigen und Messen erfolgen. Diese Möglichkeiten sind oft prohibitiv teuer. Eine intelligentere Variante ist das Verschenken von Einstiegsversionen.

A 5. Nutzer lizensieren Software

1. weil die Kosten einer Eigenentwicklung sehr hoch sind

Die Kosten für die Entwicklung einer marktreifen Software können enorm hoch sein[3]. Sie erreichen selbst bei vergleichsweise einfacher Aufgabenstellung leicht sechsstellige oder noch grössere Beträge. Um den Preis der Software zu reduzieren, müssen die Hersteller also versuchen, eine grosse Anzahl von Kopien zu vertreiben. Das ist umso schwerer, je kleiner die Zielgruppe ist. Die Zielgruppe wird kleiner, je spezialisierter und damit potentiell nutzenstiftender für den Anwender die Software ist. Das könnte ein Grund dafür sein, warum sich Tabellenkalkulations-Software auf breiter betriebswirtschaftlicher Front durchgesetzt hat, obwohl Speziallösungen für bestimmte Branchen sicher effizienter wären.

1. Beispielsweise Rechenzentrumsleiter. Diese Arten von Marketing unterliegen auch rechtlichen Verboten, z.B. des UWG für die sogenannte Kaltaquise, bei der eine Person ohne vorherigen Geschäftskontakt angerufen wird.
2. Open-Source könnte die logische Weiterentwicklung dieses Konzepts sein.
3. Vgl. kritisch zu dieser Aussage DeMarco, S. 3 ff., der dies über den Nutzen der Software relativiert: Wenn Software wirklich zu teuer wäre und der Nutzen den Aufwand für die Unternehmen nicht deutlich übertreffen würde, würden die Auftraggeber sicher nicht seit Jahrzehnten immer mehr Software haben wollen.

2. weil die Eigenentwicklung lange dauert

Wenn man von Trivialprogrammen absieht, dauern anspruchsvolle Softwareprojekte in der Regel mehrere Mannjahre[1]. Dazu muss auch bei den Auftraggebern ein entsprechendes Knowhow vorhanden sein, denn sie müssen zutreffend beschreiben können, welche Anforderungen sie an die Software stellen.

3. weil sie Standards nutzen wollen

Für einige Unternehmen und Behörden ist es wichtig, Standards zu nutzen. Davon versprechen sie sich Vereinfachungen und damit geringere Kosten[2]. Ein solcher Standard ist das DOC-Format[3] bei Textdateien. Es wird häufig verwendet, wenn zwischen Unternehmen Daten ausgetauscht werden. Dabei spielt es eine untergeordnete Rolle, ob beispielsweise Microsoft Word als Haupterzeuger[4] dieses Formats tatsächlich für die Lösung der Aufgabe angemessen ist[5]. Da die Dokumentation dieses Formates nicht frei verfügbar ist, kommt eine Entwicklung von eigener Software dafür nur selten in Frage. Bei Nutzung von Standards ergibt sich ausserdem die Möglichkeit, Software ohne weitere Anpassungen zu nutzen, die auf diesen Standards aufsetzt[6]. Ausserdem hat man die Hoffnung, dass im Falle der Einstellung des Produktes genügend andere Nutzer existieren, die dann die gleichen Probleme der Migration auf ein anderes Produkt haben, so dass sich angesichts der ausreichend grossen Installationsbasis ein Hersteller finden wird, der ein Migrationstool anbietet.

4. weil sie sich Support versprechen

Als „Belohnung“ für die Lizensierung versprechen sich die Nutzer, vom Hersteller einen privilegierten Status zu erhalten. Sie gehen davon aus, dass ihre Support-Anfragen bevorzugt behandelt werden und ihre Verbesserungsvorschläge in den folgenden Programmversionen berücksichtigt werden. Dies war zumindest Mitte der achtziger Jahre ein

1. Das Mannjahr ist eine Einheit, sie bedeutet, dass eine Person ein Jahr lang an dem Projekt gearbeitet hat. Beispiel: 50 Mannjahre, das bedeutet, dass 50 Personen 1 Jahr an der Software gearbeitet haben, oder 100 Personen ein halbes Jahr usw. Die Austauschbarkeit ist bei einer Resourcenplanung kritisch, vgl. Brooks, S. 16.
2. So hat Chrysler 1997 rund 25.000 PCs auf Microsoft Office umgestellt, weil man Inkompatibilitäten fürchtete, vgl. Hoch/Roeding/Purkert/Lindner S. 124.
3. Genauer gesagt gibt es vom DOC-Format zahlreiche Unterformate, die von den verschiedenen Versionen von Microsoft Word erzeugt werden. Diese sind weitestgehend inkompatibel zueinander, weshalb Drittprogramme regelmässig Schwierigkeiten mit der Konvertierung haben.
4. Auch andere Programme sind in der Lage, DOC-Dateien im Microsoft-Word-typischen Format zu erzeugen. Beispielsweise StarOffice/OpenOffice.
5. Häufig eignen sich Adobe PageMaker (für Flugblätter und Werbezettel) oder Adobe FrameMaker (für umfangreiche, strukturierte Dokumente) besser.
6. Beispielsweise Software, die DOC-Dateien in das PDF-Format konvertiert.

Lizensierungsargument der Hersteller. Mittlerweile ist Ernüchterung bei den Nutzern eingekehrt. Die meisten Hersteller bieten über die rechtliche Stellung als legaler Nutzer hinaus keine Vorteile durch die Lizensierung. Insbesondere gibt es kaum noch kostenlosen Support[1]. Man gewinnt den Eindruck, dass die Hersteller den Support nur als Kostenstelle betrachten[2]. Der Support setzt sich zusammen aus Fragen zur Benutzung des Programmes[3] sowie Klärung von Fehlersituationen[4]. Die Kunden verlangen im wesentlichen gar nicht, dass Software absolut fehlerfrei sein soll, nur dass der Hersteller umgehend die Fehler kostenlos behebt und ggf. proaktiv die Nutzer informiert. Der früher übliche Registrierungsprozess[5] ermöglichte dies und verdeutlichte dem Kunden die Vorteile der Lizensierung. Das ist mittlerweile die Ausnahme. Man hat eher den Eindruck, dass die Hersteller vom Kunden in Ruhe gelassen werden wollen, sobald die Kasse geklingelt hat. Die Hersteller argumentieren damit, dass Kunden, die keinen Support benötigen, die durch andere verursachten Kosten mittragen. Der durchschnittliche Nutzer wird sich eher fragen, wofür er selbst bei Massenprodukten so hohe[6] Lizensierungspreise zahlt.

5. weil sie sich rechtskonform verhalten wollen

Für bestimmte Gruppen von Nutzern ist es wichtig, sich rechtskonform zu verhalten. Das gilt beispielsweise für die Firmen, bei denen eine Person dafür haftet, dass das Unternehmen über ausreichende Lizenzen verfügt. Oder auch für Privatleute, die – wenn auch unwahrscheinlichen

1. Microsoft, Oracle und SuSE bieten lediglich Installationssupport an, der aber zeitlich oder durch die Anzahl der Anrufe beschränkt ist. Darüber hinaus sind Verträge abzuschliessen, die den Lizenzpreis für die Software deutlich übersteigen. Oft sind die Produkte professionell ohne derartige Supportverträge gar nicht nutzbar, weil man nur darüber herausfinden kann, ob der Fehler in der lizensierten Software oder der eigenen Umgebung liegt und nur so prompte Fehlerkorrekturen erhält, selbst wenn die Fehler eindeutig dem Hersteller zuzurechnen sind.
2. Obwohl der Support auf Grund seiner hohen Bedeutung für den Mehrwert des Produktes beim Kunden für den Kunden ein entscheidendes Argument für die Lizensierung ist.
3. Wichtig für die Generierung von Mehrwert durch das Programm.
4. Dabei ist oft unklar, ob es sich um Bedienungsfehler handelt, Fehler in der Umgebung des Kunden oder echte Programmfehler, die der Hersteller zu vertreten hat.
5. Das Einsenden einer Registrierungskarte mit der Anschrift des Kunden und Angaben zur Rechnerumgebung (Hardware, Betriebssystem, Einsatzgebiete).
6. Microsoft Office XP 2002 Standard kostet als Vollversion z.B. 660 Euro, http://www.softwaretrading.de; oder das Nischenprodukt ChemOffice Ultra, mehr als 100.000 Mal verkauft, 1572 Euro, also mit überschlägig 157.200.000 Euro Umsatz abzüglich Vertriebsrabatten, http://www.softline.de/home/shop/produkt/index.cfm?CFID=7425603&CFTOKEN=33956486&shopID=111&produktID=11469&info=2; oder XML Spy mit 400.000 Lizenzen, Einzellizenz $ 399, ca. $ 319,80 bei 50er-Lizenzen, also $ 127.920.000 abzüglich Rabatte, http://www.xmlspy.com/references_customers.html; Winzip, 552.462 Downloads auf http://www.download.com pro Woche, $ 29 pro Lizenz, wenn jeder 1000ste lizensiert etwa $ 833.000 pro Jahr, http://download.com.com/3101-2001-0-1.html?tag=pop.

– Ärger vermeiden möchten. Hingegen überrascht es, dass sowohl Behörden als auch Softwarehersteller selbst keineswegs immer korrekt lizensieren. Das typische Vorgehen besteht darin, eine Lizenz für einen Rechner zu kaufen, die Software aber auf mehreren Rechnern zu nutzen. Dadurch erhofft man sich, etwas „guten Willen“ gezeigt zu haben, so dass der Hersteller lediglich auf einer Nachlizensierung besteht, sofern er dies herausfindet. Unter Umständen hat man in der Zwischenzeit aber schon enorme Summen allein dadurch gespart, dass man keine Updates o.ä. bezahlt hat.

6. weil sie vom Produkt überzeugt sind

Diesen Effekt nutzen besonders die Hersteller von Shareware aus. Man kann darin Elemente des Kaufs auf Probe (§ 454 ff. BGB) sehen. Der Hersteller nimmt dabei an, dass der Interessent zum Lizenznehmer wird, sofern dieser die Software ausgiebig testen konnte und er sie nützlich findet. Oft haben diese Programme im Verhältnis zu ihrem Preis einen hohen Nutzen.

7. weil sie einen abnehmenden Grenznutzen hat

Die Devise „Viel hilft viel“ gilt auch nicht bei Software. Als plakative Aussage gilt die „80/20-Regel“. Danach sind 80 % der benötigten Funktionalität verhältnismässig einfach zu realisieren, so dass sie 20 % des Aufwandes ausmachen. Die restlichen 20 % der Funktionalität erzeugen hingegen 80 % des Aufwandes. Daher ist es auch aus Kundensicht sinnvoll, nicht zu versuchen, alles über Software zu lösen, sondern bewusst manuelle Bearbeitung für schwierig zu implementierende Funktionalität zu nutzen. Man verzichtet dann auf Individualsoftware.

8. Zusammenfassung

Nutzer lizensieren Software, weil eine Eigenentwicklung teuer ist und lange dauert, sie Standards nutzen wollen, sich Support vom Hersteller versprechen, sich rechtskonform verhalten wollen oder vom Produkt überzeugt sind. Lizensierung ist eine wirtschaftliche Alternative zur Individualsoftware, weil der Grenznutzen von Software abnimmt.

A 6. Nutzer lizensieren Software nicht

1. weil nicht-lizensierte Kopien auch funktionieren

Wie bei allen digitalen Kopiervorgängen üblich, wird die Qualität der Software durch den Kopiervorgang nicht beeinträchtigt.

Es gibt zwei Interessen des Nutzers: Einmal die Reduzierung seiner Kosten. Dies kann der Lizenznehmer dadurch erreichen, dass er für weitere

Nutzer Kopien erstellt. Er kann seine Lizenzkosten dann unter den weiteren Nutzern aufteilen[1]. Alternativ besteht das Interesse, möglichst ein Alleinstellungsmerkmal zu haben, wenn man schon Kosten hatte.

2. weil der Hersteller sowieso jede Haftung ausschliesst

Für manchen Benutzer sieht es so aus, als ob der Hersteller lediglich schnell Geld kassieren möchte, sich aber drückt, wenn es darum geht, das Produkt einem konkreten Nutzen für den Anwender zuzuführen. Die Interessen sind klar unterschiedlich: Der Anbieter einer Standardsoftware möchte möglichst nur das fertige Produkt lizensieren, aber keinen zusätzlichen Aufwand individuell für den Kunden erbringen[2]. Der Nutzer hingegen interessiert sich wenig für formale Berechtigungen durch die Lizenz[3], sondern möchte eine konkrete Problemlösung haben, die seinen Nutzen maximiert und den Lizenzpreis rechtfertigt. Die Lösung sieht für einen bedeutenden Teil der Nutzer dann so aus, dass die Software eben nicht korrekt lizensiert wird, da man aus Erfahrung weiss, dass der Hersteller sich seinerseits vor der Verantwortung drückt. Und wenn der Hersteller nicht mal für die Benutzbarkeit sorgen oder seine Fehler einstehen will, warum sollte man ihn dann noch entlohnen?

3. weil Software fehlerhaft ist

Eine Erkenntnis der Informatik ist, dass man zwar die Fehlerhaftigkeit eines Programmes leicht beweisen kann, die Abwesenheit von Fehlern dafür aber deutlich schwieriger darzulegen ist[4]. Leider ruhen sich zahlreiche Hersteller auf dieser Aussage aus und unternehmen nicht einmal die zumutbaren Qualitätssicherungsmassnahmen. Bei einigen (durchaus verbreiteten) Programmen gewinnt man den Eindruck, dass man der erste sein müsse, der jemals damit gearbeitet hat, weil sich derart eklatante Fehler darin finden. Das hat damit zu tun, dass die Qualitätssicherung in Zeiten knapper Kassen und drängender Termine gerne weggelassen wird. Insbesondere verzichtet man gerne auf das Nachtesten[5]

1. Jemand lizensiert ein Softwareprodukt (z.B. Office, Spiel) und berechnet jedem seiner Freunde 25 Euro für eine Kopie. Auf diese Weise haben alle die aktuelle Version, natürlich nicht legal, und die Ausgaben halten sich für jeden in Grenzen.
2. Es sei denn gegen üppige Vergütung. Aber selbst dann ist das Bestreben vieler Hersteller von Standardsoftware gering, denn mit zunehmenden „Extrawünschen“ würde es schwieriger, die folgenden Versionen zu pflegen.
3. Weil diese ihm allein noch keinen messbaren Nutzen verschafft.
4. Man benötigt lediglich eine einzige Eingabe, die fehlerhafte Ausgaben liefert, um die Fehlerhaftigkeit des Programmes zu zeigen. Für die Korrektheit muss man jedoch alle Eingaben (und vor allem Kombinationen von möglichen Eingaben) prüfen, und das ist aus kombinatorischen Gründen in der Praxis völlig unmöglich.
5. Nach der Korrektur eines Fehlers sind sämtliche Testfälle noch einmal zu prüfen, um auszuschliessen, dass durch die Fehlerkorrektur an anderer Stelle ein neuer Fehler eingebaut wurde.

oder automatisierte Fehleranalysen[1]. Viele Hersteller sind nicht bereit, kostenlose Fehlerberichtigungen für jeden gefundenen Fehler bereit zu stellen, insbesondere nicht unaufgefordert[2]. Sie lassen stattdessen die Fehlerberichtigungen in die neue, kostenpflichtige Version einfliessen. Das hat zur Folge, dass der Lizenznehmer für sein Geld so gut wie nie ein vollständig funktionsfähiges Programm erhält. Dazu passen auch Lizenzbestimmungen, die ausdrücklich ablehnen, irgendeine Eignung des Programmes anzuerkennen[3].

4. weil die Hardware schon teuer genug ist

Eine andere Gruppe von Nutzern lizensiert Software nicht, weil sie die verfügbaren Mittel allein oder überwiegend in die Hardware investiert. Das wäre zumindest nachvollziehbar, wenn diese gerade eben ausreichend wäre. Es lässt sich jedoch beobachten, dass eher überdurchschnittliche und nicht für produktive Hauptanwendungen[4] benötigte Hardware gekauft wird[5]. Das dafür benötigte Geld stammt allzu häufig aus dem Teil des Etats, der eigentlich für die Lizensierung von Software eingesetzt werden sollte. Besonders unter Heimanwendern lässt sich dieses Muster beobachten. Unterstützt werden sie dabei manchmal von den Verkäufern[6].

5. weil sie damit subjektiv keinen Schaden anrichten

Den Personen, die Software zwar nutzen, aber nicht lizensieren, ist häufig nicht bewusst, dass sie einen Schaden anrichten. Es gibt sogar Ver-

1. Beispielsweise durch Tools wie Mercury WinRunner, vgl. http://www-heva.mercuryinteractive.com/products/winrunner, das automatisiert die Eingaben eines Benutzers simuliert, die Ausgaben des Programmes vergleicht und somit über Nacht ausgiebig das Programm in typischen Situationen testen kann. Ebenso kann man mit speziellen Lasttest-Tools hohe Benutzerzahlen simulieren.
2. Interessant ist die Statistik des Open-Source-Web-Browsers Mozilla: So wurden allein im Februar 2002 etwa 2030 Fehlerberichte eingesandt, vgl. http://mothra.mozilla.org/webtools/miscstats. Nimmt man an, dass kommerzielle Projekte auch nicht viel weniger Fehlermeldungen erzeugen, so kann man den Aufwand allein für die Überprüfung erahnen.
3. Beim Lesen mancher Lizenzbestimmungen der Hersteller wird klar, warum der Nutzer nichts dafür zahlen möchte: Der Hersteller bestreitet deutlich, dass sein Programm überhaupt irgendeine Funktion hat, und wenn, dann nur zufällig. Es läuft auf die Formel hinaus: „Gib mir Dein Geld, dafür gebe ich Dir nichts“. Man vergleiche auch die äusserst restriktiven Microsoft-Lizenzbestimmungen (EULA, z.B. in http://www.kortstock.de/dokus/eckert/Computerrecht.pdf). Der „Endbenutzer-Lizenzvertrag für Microsoft-Software“ beim Office-Update formuliert das so: „KEINE GEWÄHRLEISTUNG. Microsoft lehnt ausdrücklich jede Gewährleistung für die ADD-IN-KOMPONENTEN ab. Die ADD-IN-KOMPONENTEN und die dazugehörige Dokumentation werden ’wie besehen’ zur Verfügung gestellt, ohne Gewährleistung jeglicher Art, sei sie ausdrücklich oder konkludent, einschliesslich, jedoch nicht beschränkt auf jede konkludente Gewährleistung im Hinblick auf Handelsüblichkeit, Eignung für einen bestimmten Zweck oder Nichtverletzung von Rechten Dritter. Das gesamte Risiko, das sich aus der Verwendung oder der Leistung der ADD-IN-KOMPONENTEN ergibt, liegt bei Ihnen. KEINE HAFTUNG FÜR SCHÄDEN. […]“.
4. Beispielsweise Textverarbeitung für Hausarbeiten.
5. Z.B. teure Grafikkarten oder DVD-Laufwerke, die für Spiele benötigt werden.

treter dieser Gruppe, die vehement behaupten, die Hersteller mögen sich freuen, dass ihre Software eingesetzt wird und somit Verbreitung findet[1]. Gerne wird auch argumentiert, der Softwareindustrie sei kein Schaden entstanden, weil man sich die Software sowieso nicht hätte leisten können.

6. weil es keinen wirksamen Kopierschutz gibt

Seit etwa 20 Jahren versucht die Industrie, das Problem der verlustfreien und einfach zu erstellenden digitalen Kopie zu lösen. Dabei kamen beispielsweise Dongles, Passwörter und Verschlüsselungsmassnahmen zum Einsatz.

a) Hardwarebindung

Ein einfaches Beispiel für die Bindung von Software an eine bestimmte Hardware ist die Abfrage auf die Existenz eines Dongles[2]. Man kann dabei so vorgehen, dass man im Binärcode des Programmes die entsprechende Stelle sucht, die die Abfrage initiiert, und genau davor eine Sprunganweisung einfügt, mit der die Abfrage einfach übersprungen wird. Der Rest des Programmes läuft, saubere Manipulation vorausgesetzt, problemlos.

b) Kennwörter

Es ist auch versucht worden, die Software durch notwendige Eingabe von herstellervergebenen Kennwörtern z.B. beim Programmstart zu schützen[3]. Einerseits können diese Kennwörter zumeist einfach weitergegeben werden. Andererseits ereilt die Abfrage der Kennwörter meist das gleiche Schicksal wie die Abfrage eines Dongles: die Abfragen werden durch Manipulation übersprungen.

c) Verschlüsselung

Man kann auch den gesamten Programmcode verschlüsseln, so dass er nur in einer bestimmten Umgebung[4] beim Starten dynamisch entschlüsselt werden kann. Nun lassen sich zwar keine Abfragen mehr überspringen. Jedoch kann spätestens durch Beobachten des Prozessors oder Ab-

6. So bot der Abteilungsleiter eines grossen Kaufhauses einem Kunden im Sommer 2001 an, er könne Software auf dem Apple-Rechner „vergessen zu deinstallieren“, damit der Umstieg vom PC nicht so teuer werde.
1. Z.B. http://www.heise.de/newsticker/foren/go.shtml?read=1&msg_id=1375082&forum_id=24809.
2. Der Hersteller liefert das Dongle zusammen mit der Software. Es handelt sich um einen Stecker mit einem Chip, der auf eine Schnittstelle (seriell, parallel, USB) gesteckt werden muss. Die Software lässt das Dongle bestimmte Werte berechnen und läuft nicht mehr, wenn falsche oder gar keine Werte geliefert werden, vgl. auch Schricker/Loewenheim § 69 f Rn 9.
3. Vor einigen Jahren wurden Computerspiele so vertrieben. Dem Handbuch lag eine Liste mit Passwörtern bei, die reihum abgefragt wurden.
4. Zurückgelieferte Dongle-Werte, Mainboard-Seriennummer, Kennwörter etc.

laufenlassen in einem Emulator die unverschlüsselte Folge von Anweisungen ermittelt werden. Diese kann in eine Datei geschrieben und im folgenden statt der verschlüsselten Version eingesetzt werden.

d) Haltbarkeitsdatum

Einige Programme sind mit einem „Haltbarkeitsdatum“ versehen, so dass sie nur bis zu einem bestimmten Datum funktionieren. Dieses Datum kann entweder vom Hersteller fest eingesetzt[1] oder von der ersten Installation abhängig sein[2]. Durch Verstellen des Datums kann man entsprechend wieder Zugang zum Programm erhalten. Die Verwender dieser Schutzmassnahme setzen darauf, dass es unbequem ist, dauerhaft die Systemzeit falsch einzustellen. Zudem ist das Zurückstellen unpraktikabel, sofern man mehrere Programme nutzen möchte, die auf diese Weise geschützt sind, oder das Datum zwingend benötigt (weil Daten mit diesem Datum abgelegt werden, häufig bei primitiven Buchhaltungs- oder Versionskontrollsystemen der Fall).

e) Nag-Screens

Mit genau dieser Zielsetzung gibt es z.B. bei Shareware die sogenannten Nag-Screens[3]. Das sind Fenster, die beim Programmstart erscheinen und den Benutzer zur Lizensierung auffordern. Die Programme enthalten einen Zähler, der angibt, wie oft oder an wie vielen Tagen das Programm bereits gestartet wurde. Teilweise werden solche Fenster auch im Abstand von z.B. 15 Minuten eingeblendet. Sie sollen einfach nur bewirken, dass der Nutzer sich „ärgert“, keine lizensierte Version zu haben. Dabei bleibt die Software durchaus benutzbar. Anwender, die das Programm eine Weile getestet haben und gut finden, registrieren es und zahlen an den Hersteller. Diejenigen, die nicht zahlen, aber es trotzdem nutzen, hätten sonst mit hoher Wahrscheinlichkeit einen etwaigen Kopierschutz ausgehebelt bzw. sich solch eine Version besorgt. Da man die Software grundsätzlich noch nutzen kann, spekulieren die Nag-Screen-Verwender darauf, dass sich niemand die Mühe macht, diesen Schutz auszuhebeln. Zudem sind derartige Shareware-Programme meist in Preisregionen, die man als preisgünstig bezeichnen kann[4].

f) Zusammenfassung

Letztlich bieten keines der genannten Verfahren einen wirksamen Ko-

1. Vor dem jeweiligen Download in das Programm hinein generiert.
2. Legt eine Datei auf dem System an, in der das Datum steht.
3. Vgl. http://spiderwebsoftware.com/shareware/limiting.html mit weiteren Methoden der Limitierung von Shareware.
4. Die meisten dürften bei 50 Euro liegen, selten geht es bis 500 Euro.

pierschutz. Dafür gibt es mehrere Gründe. Einer davon ist, dass schon eine einzige manipulierte Kopie ausreicht, um davon verlustfrei beliebig viele illegale Kopien erstellen zu können. Je begehrter die Software, desto schneller steht in der Regel so eine Version zur Verfügung. Zur Zeit können Kopierschutzsysteme daher keinen absoluten Schutz bieten. Sie erschweren es dem Anwender nur. Problematisch ist, dass die Hersteller durch einige Kopierschutzmassnahmen die Nutzer derart behindern, dass diese selbst als Lizenznehmer lieber eine manipulierte Version einsetzen, um sich den Ärger z.B. mit Hardwareinkompatibilitäten zu ersparen. Das könnte sich ändern, sobald keine Rechner ohne umfassenden DRM-Support[1] mehr auf dem Markt erhältlich sind.

7. weil Urheberrechtsverstösse kaum verfolgt werden

Urheberrechtsverstösse sind mittlerweile zum Volkssport geworden. Ob es sich um Software oder Content (Musik, Filme) handelt, der Durchschnittsbürger hat kein Problem damit, keine rechtmässigen Lizenzen zu besitzen. Das liegt einerseits an der Information und Aufklärung. Den meisten Menschen dürfte die Funktionsweise und Zielsetzung des Urheberrechts nicht bewusst sein. Gleichzeitig hat die Industrie erreicht, dass die Werke immer wichtiger werden, weil sonst „etwas im Leben fehlt“. Da in der Praxis aufgrund der seltenen Aufdekkung keine Strafverfolgung oder Schadenersatzansprüche zu befürchten sind, wird kopiert so viel und so lange es geht. Davon zeugen die zahlreichen „Tauschbörsen“ und „Warez-Seiten“ im Internet.

8. weil sie sich subjektiv die Lizenzen nicht leisten können

Zunächst denkt man dabei an die „üblichen Verdächtigen“ mit niedriger Kaufkraft: Schüler, Studenten, Auszubildende, Arbeitslose und Sozialhilfeempfänger. Bei näherer Betrachtung stellt sich jedoch heraus, dass auch grosse Firmen und Behörden sich die Lizenzkosten subjektiv nur schwer leisten können[2]. Subjektiv deshalb, weil sie objektiv die Möglichkeit hätten, entweder die Lizenzkosten aufzubringen oder eben auf das Produkt zu verzichten[3]. Letztere Alternative wird in der Praxis selten genutzt, u.a. weil man unbedingt aus Kompatibilitätsgründen einen

1. Dazu muss die gesamte Hardware und Software DRM unterstützen. Beispielsweise sendet die Grafikkarten ihre Daten verschlüsselt zum Monitor, um ein Abgreifen der Videosignale zwecks Erstellung einer illegalen Video-Kopie zu verhindern. Oder die Festplatte weigert sich, eine Musikdatei abzuspeichern, weil dieser keine gültige Lizenzerklärung beigefügt ist.
2. Der Grund liegt in der grossen Anzahl von benötigten Lizenzen. Die Hersteller bieten deshalb in der Regel Sonderkonditionen, aber für eine Behörde mit 500 oder 70.000 Arbeitsplätzen ergeben sich absolut gesehen dennoch hohe Zahlen.

Marktführer einsetzen möchte. Hinzu kommen Zweifel, ob die Software betriebswirtschaftlich tatsächlich ein ausreichendes Return On Investment (ROI) bietet[1]. Auch lassen sich die Total Cost of Ownership (TCO) schwer bestimmen und werden daher oft völlig ignoriert[2].

Allerdings kann auch die schiere Anzahl von zu lizensierenden Programmen pro Rechner in der Summe dazu führen, dass die Lizenzen unerschwinglich sind. Für den Büro-PC einer kleinen Einmann-Firma kann man leicht Software im Wert von mehreren tausend Euro benötigen[3], zuzüglich der jeweiligen periodischen Updates[4].

9. weil sie vom Hersteller enttäuscht sind

Einige Hersteller scheinen es mit ihrer Fahndung nach „Raubkopierern" zu übertreiben. So sprechen einige bereits von einem „Lizenzkrieg mit Microsoft"[5]. Die Firma Borland liess sich in den Lizenzbedingungen sogar das Recht zu Hausdurchsuchungen beim Kunden zur Feststellung von Minderlizensierungen einräumen[6]. Man verärgert so treue Kunden, indem man sie mit Verdächtigungen und hohen Lizenzkosten überzieht. In der Folge vermeiden die Unternehmen jede Lizensierung, sofern möglich und suchen nach einer günstigen Alternative.

10. weil sie den Hersteller für reich genug halten

Einige Nutzer lizensieren Software nicht, weil sie den Hersteller für reich genug halten. So wird z.B. Spielesoftware häufig illegal kopiert. Dennoch machten die Urheber im Jahr 2001 einen Umsatz von 2,9 Mil-

3. Als Analogie mag der Diebstahl eines teuren Autos dienen, mit der Rechtfertigung, dass man sich zu Fuss nur sehr unbequem und langsam fortbewegen könne. Dabei hinkt die Analogie, weil die Original-Software im Gegensatz zum Auto dadurch nicht an Nutzbarkeit verliert. Jedoch wird das Bestimmungsrecht des Urhebers missachtet und die Software verliert dadurch an Wert, dass legale Lizenznehmer trotz Zahlung keinen angemessenen Vorteil gegenüber Nicht-Lizenznehmern mehr haben.
1. Dazu müssten durch die Verwendung einer Software mindestens Aufwendungen in Höhe ihres Lizenzpreises eingespart werden. Daran gibt es oft berechtigte Zweifel. So sieht der heutige Geschäftsbrief vielleicht besser aus als vor 20 Jahren, dies schlägt sich aber nicht in ersparten Aufwendungen nieder. Eine ROI-Rechnung für Software ist deshalb schwer durchzuführen.
2. Dazu gehören u.a. auch die Kosten für die Hardwareaufrüstung, Administration, Schulung, aber auch Effizienzveränderungen, vgl. „ROI und TCO machen die IT-Kosten transparent", COMPUTERWOCHE Nr. 08 vom 22.02.2002, http://www.computerwoche.de/index.cfm?pageid=267&type=ArtikelDetail&id=80106547&cfid=4796566&cftoken=31071324&nr=7.
3. Windows XP Home 236 Euro, Office XP Professional 769 Euro, Adobe Web Collection 1599 Euro, Adobe Acrobat 359 Euro, Lexware Financial Office 248 Euro, Norton Internet Security 79 Euro, WinZip 49 Euro, zusammen etwa 3400 Euro, http://www.softline.de und http://www.softwaretrading.de.
4. Updates werden häufig irrational durchgeführt. Es wird selten berechnet, welchen Nutzen das Update hat. So sind die wesentlichen Funktionen von Microsoft Word seit über 15 Jahren gleich geblieben, während rund 10 Updates dazwischen liegen. Entscheidend ist, dass z.B. Geschäftspartner Dokumente im neuen Datenformat senden könnten und dieses weitgehend inkompatibel zum alten Format wäre.
5. AGEV-Magazin 4/2002, S. 5, http://www.agev.de; vgl. zu den umstrittenen Windows XP-Lizenzen auch http://www.heise.de/newsticker/data/hps-11.02.02-000.

liarden Mark[1]. Für die Ersteller und Nutzer illegaler Kopien ist es daher schwer einsehbar, warum den Urhebern ein relevanter Schaden entstehen soll. Ein häufiges Argument ist, dass man zwar einen normalen Lebensstil der Entwickler in Ordnung findet, aber nicht helfen möchte, ihnen ein Leben in Luxus „auf Kosten der Nutzer“ zu ermöglichen. Dabei werden häufig die enormen Kosten der Softwareentwicklung übersehen oder schlicht Umsatz mit Gewinn verwechselt.

11. Zusammenfassung

Es gibt viele Gründe, warum Nutzer die Software nicht lizensieren, obwohl sie benutzt wird. Nicht lizensierte Software erfüllt ebenso ihren Zweck und ist leicht zu bekommen. Ihr Einsatz wird kaum ernsthaft verfolgt. Als Rechtfertigung wird darauf verwiesen, dass man sich die Software sowieso nicht hätte leisten können, also gar kein Schaden enstanden und die Hersteller reich genug seien. Zudem sei die Software eh fehlerhaft und der Hersteller drücke sich vor jeder entlohnenswerten Verantwortung.

A 7. Allgemeine Tatsachen

1. Nicht kommerzielle Urheber entwickeln Software

Darunter fallen Privatpersonen ebenso wie Vereine und öffentliche Forschungseinrichtungen[2]. Kennzeichnend für diese Gruppe ist, dass keine Gewinnerzielungsabsicht vorliegt. Die nicht kommerziellen Urheber gehen davon aus, durch die Erschaffung der Software keine Gewinne zu erzielen. Es ist auch möglich, dass sie nur subjektiv davon ausgehen, dass auf Grund der Situation eine Gewinnerzielung nicht möglich ist (obwohl objektiv möglich). Ein häufiges Motiv ist die Kostenvermeidung, d.h. man lizensiert kein existierendes Programm, sondern entwikkelt ggf. in der Freizeit ein eigenes, um Lizenzgebühren zu sparen[3].

a) Privatpersonen

Zu den Privatpersonen gehören Schüler und Studenten. Beide Gruppen fallen dadurch auf, dass sie gegenüber Arbeitnehmern über verhältnismässig viel frei einteilbare Zeit verfügen. Häufig sind sie auch nicht gezwungen, wirtschaftlich auf eigenen Beinen zu stehen (beispielsweise weil die Eltern sie noch unterstützen). Daher haben Schüler und Studen-

6. AGEV-Magazin 4/2002, S. 5, http://www.agev.de; http://www.heise.de/newstikker/data/hes-14.01.02-000, später teilweise zurückgenommen: http://www.heise.de/newsticker/data/hes-16.01.02-000.
1. Vgl. http://www.spiegel.de/netzwelt/netzkultur/0,1518,194943,00.html.
2. Die BMBF-Studie ignoriert diese Urheber.
3. Der Zeitbedarf wird von vielen Menschen nicht zu den Kosten gezählt, da sie in der fraglichen Zeit keine Einkünfte erzielen können.

ten verhältnismässig viel Zeit, um in ihrer Freizeit Software zu entwikkeln. Bei Privatpersonen kann das Motiv in einem Hobby begründet sein. Bei Studenten der Informatik oder ähnlicher Fächer kommt hinzu, dass sie im Rahmen des Studiums Software entwickeln müssen. Die berühmtesten software-entwickelnden Schüler dürften die Entwickler von GIMP sein[1]. Einer der berühmtesten Studenten dürfte Linus Torvalds sein, der Begründer des Open-Source-Betriebssystems Linux[2].

b) Vereine

Vereine entwickeln Software für interne Zwecke (z.B. Mitgliederverwaltung, Wettkampfstatistiken) oder stellen sie ihren Mitgliedern zur Verfügung. Es gibt auch Vereine, die Software zur allgemeinen Verbreitung entwickeln. Als Beispiel kann der CCC[3] dienen.

c) Staatliche Forschungseinrichtungen

Eine Reihe von staatlichen Forschungseinrichtungen[4] entwickeln Software. Vornehmlich handelt es sich dabei um die Universitäten, die Software für den Eigenbedarf in Forschung und Lehre, aber auch ihrer eigenen Verwaltung, entwickeln.

d) Öffentliche Verwaltung

Auch die öffentliche Verwaltung entwickelt in grossem Umfang Software. So wird beispielsweise von den Finanzämtern das Projekt Fiscus[5] vorangetrieben, eines der grössten IT-Projekte in Europa überhaupt. Das Bundeskriminalamt entwickelt Inpol Neu[6]. Die Bundesanstalt für Arbeit[7], die Versorgungsanstalt des Bundes und der Länder[8] und die Landesversicherungsanstalten[9] entwickeln ebenfalls Software.

e) Zusammenfassung

Eine bunte Sammlung nicht kommerzieller Urheber entwickelt Soft-

1. Vgl. http://www.gimp.org oder http://www.acm.org/crossroads/xrds3-4/gimp.html.
2. Wobei dies korrekterweise GNU/Linux heissen müsste, da es einen nennenswerten Anteil von Software aus dem Projekt GNU benötigt (http://www.gnu.org).
3. Chaos Computer Club, vgl. http://www.ccc.de.
4. Ausgeschlossen werden Forschungseinrichtungen innerhalb von Unternehmen.
5. Software für die Finanzverwaltung, z.B. „Bussgeld- und Strafsachenstelle“ sowie „Festsetzung Grunderwerbsteuer“, 120.000 Anwender in 700 Finanzämtern mit 30 Millionen Steuerfällen pro Jahr, mittlerweile in Form einer GmbH mit über 200 Mitarbeitern: http://www.fiscus.info; man schätzt die Kosten auf 1,4 Milliarden DM, vgl. http://www.spiegel.de/spiegel/0,1518,141491,00.html und Generalanzeiger Bonn vom 11.04.2002.
6. Zentrale Auswertung mehrerer Datenbanken, vgl. http://www.heise.de/newstikker/data/jk-04.01.01-005; man spricht von einem 80 Millionen-Flop: vgl. http://www.spiegel.de/politik/deutschland/0,1518,177264,00.html; andere schätzen die Kosten auf 240 Mio. DM, http://www.heise.de/newsticker/data/jk-18.02.01-004.
7. Internet-Abfrage für Stellenangebote, http://www.arbeitsamt.de.
8. Ein „IT-Desaster“ für über 100 Millionen DM, vgl. http://www.heise.de/newstikker/data/jk-06.12.01-001.
9. Abfrage der Rentenversicherungsdaten per Internet, Kooperation zwischen http://www.lva.de und http://www.db24.de.

ware, darunter Privatpersonen, Vereine, Forschungseinrichtungen und die öffentliche Verwaltung.

2. Kommerzielle Urheber entwickeln Software

Kommerzielle Urheber sollen hier über ihre Gewinnerzielungsabsicht definiert werden. Software erstellende Unternehmen lassen sich in eine Primärbranche und eine Sekundärbranche unterteilen.

a) Primärbranche

Zur Primärbranche gehören Hardwareberatungen, Softwarehäuser (Softwareberatung, Softwareentwicklung), Datenverarbeitungsdienste (Datenerfassungsdienste, Tabellierungsdienste) und Anbieter von Datenbanken sowie die Hersteller von Datenverarbeitungsgeräten und -einrichtungen[1], soweit sie jeweils selbst Software entwickeln[2]. Aus diesen Kriterien ergeben sich in Deutschland etwa 10.568 Unternehmen[3]. Davon haben 77 % nur 1-9 Mitarbeiter, 16 % nur 10-49 Mitarbeiter[4]. Die durchschnittliche Mitarbeiterzahl liegt bei 28, davon 12 Softwareentwickler[5]. Überwiegend handelt es sich also um kleine Betriebe. Im Durchschnitt sind diese Unternehmen 10 Jahre alt[6].

Diese Unternehmen möchten mit einer Software als Hauptprodukt ihre Umsätze erzielen. Zu den grösseren Unternehmen dieser Gruppe gehören mit Schwerpunkt in den USA die Softwarefirmen wie Microsoft[7] oder Oracle[8], mit Schwerpunkt in Deutschland beispielsweise SAP[9], IDS Scheer[10], Software AG[11], SuSE Linux AG[12] oder die Ixos Software AG[13]. Im Kern sind diese Unternehmen auf ein Software-Produkt fokussiert, während sie ggf. auch damit zusammenhängende Hardware und Dienstleistungen anbieten.

Die Primärbranche läßt sich weiterhin unterscheiden nach der Anzahl der Installationen pro Version der Software: Individualsoftware (<100), Kleinserien (100 bis 1.000) und Standardsoftware (>1.000)[14]. Dabei entwickeln 45 % der Unternehmen der Primärbranche Individualsoft-

1. BMBF S. 31.
2. Das macht nur etwa ein Drittel der Unternehmen, BMBF S. 56.
3. BMBF S. 57.
4. BMBF S. 59.
5. BMBF S. 59.
6. BMBF S. 61.
7. Betriebssysteme/Office/Entwicklungswerkzeuge, http://www.microsoft.de.
8. Datenbanksysteme/Application Server, http://www.oracle.de.
9. Betriebswirtschaftliche Branchenlösungen, http://www.sap-ag.de.
10. Geschäftsprozessmanagement, vgl. http://www.ids-scheer.de.
11. Datenbanksysteme/Middleware, http://www.softwareag.de.
12. Linux-Distribution und -Lösungen, http://www.suse.de.
13. Dokumentenmanagement, http://www.ixos.de.
14. BMBF S. 72.

ware, 32 % Kleinserien und 38 % Standardsoftware[1]. Zu 86 % ist die Software ein eigenständiges Produkt, 24 % wird als Teil anderer Softwareprodukte verwendet (Bibliotheken, Module, Treiber etc.) und 15 % ist Teil anderer Produkte (z.B. Maschinen, Elektronik)[2].

Auf betriebswirtschaftliche Software entfallen 55 %, Multimedia/Internet 46 %, technische Software 33 %, Systemsoftware 17 % und sonstige Software 36 %[3]. 2.900 Unternehmen bieten Software im Bereich Auftragsabwicklung/Fakturierung an, 2.400 im Materialwirtschaft/Lager-/Bestellwesen und 2.300 im Finanz-/Rechnungswesen[4]. 1.400 Anbieter liefern Software für technische Berechnungen, 1.300 im Bereich Mathematik/Statistik/Simulation, 1.100 CAD/CAM/CAE-Systeme[5]. Immerhin noch 1.000 Unternehmen liefern Systemsoftware zum Bereich Datenbanksysteme/Datenverwaltung, 900 zu Netzwerken/Rechnerkommunikation und 640 für Systemadministration/Rechenzentrums-Management[6].

Der Umsatz der Primärbranche lag 1999 bei 83 Milliarden DM, das sind pro Unternehmen durchschnittlich 8 Millionen DM[7]. Die meisten Unternehmen (54 %) erzielten 1 bis 4,9 Millionen DM Umsatz, weitere 31 % begnügten sich mit weniger als 1 Million DM[8].

Etwa die Hälfte aller Unternehmen generieren mit der Softwareentwicklung über 75 % ihres Umsatzes[9].

Die Bruttowertschöpfung der Softwareentwicklung liegt bei 37,28 Milliarden DM, das sind 1,03 % der Gesamtbruttowertschöpfung[10].

76 % der Unternehmen entwickeln Software ausschliesslich im Inland[11].

b) Sekundärbranche

Die Sekundärbranche umfasst alle Unternehmen, die Software als Teil von Produkten und/oder Dienstleistungen bei der Produktion, Planung oder dem Management einsetzen[12]. Letztlich zählen dazu alle Bereiche

1. BMBF S. 73.
2. BMBF S. 66.
3. BMBF S. 67.
4. BMBF S. 68. Die Zahlen verwundern auf den ersten Blick. Intuitiv würde man wohl gerade im Finanz- und Rechnungswesen sehr ähnliche Produkte in diesen Bereichen und damit deutlich weniger Anbieter erwarten.
5. BMBF S. 69.
6. BMBF S. 70.
7. BMBF S. 76.
8. BMBF S. 75.
9. BMBF S. 83.
10. BMBF S. 82, zum Vergleich: Land-, Forstwirtschaft und Fischerei haben zusammen gerade einmal 1,18 %.
11. BMBF S. 86.
12. BMBF S. 33.

von Wirtschaft und Verwaltung[1], wobei in der BMBF-Studie nur 8.660 Unternehmen der Branchen Maschinenbau, Elektrotechnik, Fahrzeugbau, Telekommunikation sowie Finanzdienstleistungen ausgewählt wurden[2]. Davon hatten 47 % nur 1-9 Mitarbeiter, 26 % nur 10-49 Mitarbeiter. Jedoch liegt die durchschnittliche Mitarbeiterzahl mit 287 erheblich höher als in der Primärbranche, wovon allerdings nur 7 Softwareentwickler sind[3]. Die Unternehmen der Sekundärbranche sind durchschnittlich 38 Jahre alt[4].

Die Unternehmen entwickeln zu 74 % Individualsoftware, 26 % Kleinserien und nur 6 % Standardsoftware[5]. Die Software wird zur Veredelung von Produkten[6] oder zum „Enablen“ genutzt.

Hierbei wird immerhin bei 53 % aus der Software ein eigenständiges Produkt, bei 18 % ein Teil anderer Softwareprodukte und zu 47 % ein Teil anderer Produkte[7].

Die durchschnittlichen Umsätze sind erheblich höher als die in der Primärbranche. Sie reichen von 31 Millionen DM (Maschinenbau) bis hin zu 549 Millionen DM (Finanzdienstleister)[8]. Jedoch sind damit die Gesamtumsätze gemeint, so dass sich nicht direkt auf die durch Software generierten Umsätze schliessen lässt. Dazu wird angegeben, dass durchschnittlich 15 % der Neuentwicklungskosten von Produkten und Dienstleistungen auf die Softwareentwicklung entfallen[9]. In der Telekommunikation sind es 23 %, in der Elektrotechnik 25 %, bei Telefonvermittlungssystemen sogar bis zu 80 %[10].

Durchschnittlich werden etwa 11 % des Umsatzes durch Software generiert, wobei diese Einschätzung als zu gering angesehen wird[11].

Die Bruttowertschöpfung beträgt 12,73 Milliarden DM und damit 0,35 %[12]. In vielen Branchen wird die Meinung vertreten, dass ohne entsprechende Software überhaupt keine Umsätze mehr erzielbar wären[13].

1. BMBF S. 33.
2. BMBF S. 33, 57. Die Branchen Chemie/Pharma, Transport/Verkehr, Handel und Medien wurden quantitativ nicht berücksichtigt, vgl. S. 33.
3. BMBF S. 59.
4. BMBF S. 61.
5. BMBF S. 73.
6. BMBF S. 46.
7. BMBF S. 66.
8. BMBF S. 77.
9. BMBF S. 78.
10. BMBF S. 78.
11. BMBF S. 83 f.
12. BMBF S. 82.
13. BMBF S. 46.

Ausschliesslich im Inland entwickeln 79 % der Unternehmen[1].

c) Zusammenfassung

Die Unternehmen der Primär- und Sekundärbranche erzielen eine relativ hohe Wertschöpfung durch die Entwicklung von Software.

3. Entwickler sind abhängig von Zulieferern

Immerhin 37 % der Unternehmen haben bei einzelnen oder mehreren Auftraggebern den subjektiven Eindruck, dass sie auf den Zulieferer von Software über ein normales Maß hinaus angewiesen sind[2]. 17 % fühlen sich von einzelnen Auftragnehmern und 20 %, besonders kleine Unternehmen der Sekundärbranche, von mehreren Auftragnehmern abhängig[3]. 87 % dieser Auftragnehmer haben ihren Sitz im Inland, 44 % in Nordamerika und 21 % in der Europäischen Union[4].

4. Softwareentwickler sind kreative Menschen

Etwas deutlicher lautet die These: Softwareentwickler sind Künstler. Man kann sich darüber streiten, ob sie nicht Ingenieure sind oder zumindest sein sollten. In der Praxis zeigt sich jedoch, dass weder mit Fleiss noch Handwerkszeug allein kaum ein Projekt realisiert werden kann. Oft bedarf es guter Ideen und innovativer Ansätze, die sich (noch) nicht in Büchern finden lassen. Die Leistungsmessung bei Softwareentwicklern ist daher ebenso problematisch wie die von Künstlern. Man kann letztlich nur feststellen, ob das Werk fertig ist und dem Kunden gefällt. Ob der Entwickler dafür täglich 8 oder 12 Stunden benötigt hat, ist jedoch nebensächlich für diese Leistung. Oftmals versucht das Management in Softwareprojekten Leistungsanreize zu bieten, die im typischen Produktionsbranchen üblich sind. Gerade an derartigen Akkordlöhnen zeigt sich die Absurdität der Annahme, dass sich dadurch Software in Quantität und Qualität verbessern liesse. Unter erhöhten Mengenvorgaben leidet (bei gleichem Resourceneinsatz) direkt die Qualität.

Zu einem guten Stück ist der Softwareentwickler auch Erfinder. Wenn ihm keine gute Idee kommt, kann er sich zwar (beispielsweise über Literatur oder Brainstorming mit Kollegen) motivieren, aber die Zeit bis zu einer guten Idee ist nicht sinnvoll abschätzbar. Viele Projektmanager und Auftraggeber verkennen das. Eine der wesentlichen Stärken von Open Source, nämlich ihre Stabilität und hohe Sicherheit, resultieren

1. BMBF S. 86.
2. BMBF S. 84.
3. BMBF S. 84.
4. BMBF S. 85, Mehrfachnennungen waren möglich.

daraus, dass das Produkt eben fertig ist, wenn es fertig ist. Bei Open Source erhält man typischerweise nie die Aussage „Am 1.7. erscheint die neue Version“.

5. Direktbeziehung zwischen Urheber und Nutzer ist ideal

Es ist oft für beide Seiten förderlich, wenn der Softwarehersteller zum Nutzer der Software einen direkten Kontakt hat. Für den Hersteller ist es leichter, die Anforderungen des Kunden zu ermitteln und auf ihn einzugehen, da er z.B. Informationen über die betrieblichen Abläufe des Kunden erhalten kann. Der Nutzer hat einen Vorteil, weil er gut auf ihn abgestimmte Software erhält und im Fehlerfall schnell Korrekturen verfügbar sind. Auch finanziell kann sich diese direkte Kopplung für beide Seiten lohnen, da Zwischenhändler ausgeschlossen werden. Der Nachteil liegt klar im Verzicht auf Arbeitsteilung. Typischerweise kann der Hersteller nicht so viele Kunden bedienen, wie ein Distributor. Ausserdem ist der Hersteller gezwungen, selbst Werbung, Versand und das Inkasso zu übernehmen.

6. Dienstleister leben von schlechter Software

Fehlerhafte und schwer zu verstehende Software ist ein Garant für Auslastung und gute Bezahlung bei Dienstleistern, z.B. im Bereich Installation, Administration, Schulung und Support. Diese sind nahezu unersetzbar, sofern sie die Software dennoch in den Griff bekommen und Workarounds zur Vermeidung der Probleme gefunden haben. Wenn die Software „keinen Spass macht“, können sich die Dienstleister auch für ihre hohe Frustrationstoleranz bezahlen lassen. Bei fehlerfreier und einfach zu verstehender Software sind z.B. Administratoren entbehrlich. Gleiches gilt für Schulungen, Fachbücher und Support-Hotlines.

7. Fehlerhotline ist teuer

Sofern ein Hersteller eine Hotline unterhält, über die Fehler gemeldet werden können, ist dies ein teures Unterfangen. Es ist schwer zu unterscheiden, ob es sich um Bedienungsfehler oder echte Programmfehler handelt. Zum Teil versuchen die Hersteller deshalb an solchen Hotlines dem Kunden zu erklären, dass es sich um einen Bedienungsfehler handele, obwohl es ein Programmfehler ist. Das hat zwei Ursachen. Zum einen ist das Aufspüren und Reproduktion eines Fehler schwierig. Die Systeme unterscheiden sich – trotz gemeinsamen Betriebssystems – sehr stark[1]. Von der Fehlerbeschreibung durch den Benutzer bis zu einer wiederholbaren Anleitung, wie man den Fehler immer wieder er-

zeugen kann, vergehen Stunden und manchmal Tage.

Zum zweiten müsste der Hersteller dann eingestehen, dass es sich um einen Fehler in seinem Produkt handelt. Das ist einmal aus rechtlicher Sicht nicht gerade vorteilhaft. Das Bestreiten von Fehlern bietet rechtlich mehr Möglichkeiten. Andererseits trägt der Hersteller leicht einen Image-Schaden davon, wenn er seine Fehler öffentlich zugibt und veröffentlicht[1]. Ein wesentliches Element des Capability Maturity Models ist, dass Probleme gelöst, und nicht verschwiegen werden. Man sehe sich die Anzahl der Hersteller an, die nur auf Level 1 sind[2].

8. Schad-Software generiert Softwareumsatz

Ein Phänomen ist Anti-Virensoftware: Diese Software wird entwickelt, um Schadprogramme zu entdecken und deaktivieren. Ihre Hersteller profitieren jedoch von neuen Viren, so dass schon häufig der Verdacht geäussert wurde, die Hersteller würden selbst diese Viren in Umlauf bringen, um das passende Gegenmittel verkaufen zu können.

Ähnlich ist es mit Sicherheitssoftware, besonders für das Internet. Ihre Anbieter profitieren, wenn es viele Einbrüche in Datensysteme gibt. Besonders im Internet ist es leicht, derartige Einbrüche zu provozieren.

Diese Anbieter profitieren grundsätzlich von einer FUD[3]-Strategie. Es lohnt sich also, dem Kunden Angst zu machen. Ähnlich wurde im Nachhinein auch die Jahr-2000-Problematik bewertet, wobei das unentscheidbar ist[4].

---

1. Ein Sprichwort der IT-Folklore sagt: Es ist unmöglich im Umkreis von 100 km zwei Rechner zu finden, die exakt (!) identisch sind. Damit ist weniger die BIOS-Seriennummer oder die MAC-Adresse der Netzwerkkarte gemeint, als vielmehr die unterschiedlichen Geräteeinstellungen (z.B. Druckertreiber) und Benutzereinstellungen (grosse/kleine Bildschirmschrift, Auflösung usw.).

1. Die Firmen IBM, Sun Microsystems und Microsoft unterscheiden sich stark in diesem Punkt. IBM hat lange Zeit kostenlos und zeitnah Fehlerberichtigungen veröffentlicht, und zwar auch dann noch, wenn das Produkt selbst bereits durch eine kostenpflichtige Nachfolgerversion ersetzt war. Sun Microsystems führt für die Programmiersprache Java und zugehörige Bibliotheken eine sogenannte Bug-Parade (http://www.javasoft.com). Darüber können Entwickler Fehler selbst eingeben und den Status der Evaluierung verfolgen. Die Entwickler können auch eine Präferenzliste erstellen, welche Fehler für sie am dringlichsten zu beheben sind. Ähnlich verfährt Sun mit Verbesserungsvorschlägen. Die BugParade ist nach einer kostenlosen Registrierung frei zugänglich. Microsoft hingegen bietet keinen solchen Service. Hier hat der Kunde nur Zugang zu kostenlosen Bugfixes. Allzu häufig wird jedoch kritisiert, dass die eigentlichen Schwachstellen und Fehler erst in der Folgeversion behoben werden, für die der Hersteller Geld haben möchte.
2. Vgl. „3. muss Qualität durch den Prozess sicherstellen" auf Seite 28.
3. Fear, Uncertainty, Doubt. Damit ist gemeint, dass Ängste geschürt werden, Unsicherheiten geschaffen und Zweifel kommuniziert werden. Wenn der Konsument verunsichert ist, bevorzugt er den – vermeintlich sicheren – Marktführer und gibt für Produkte mehr Geld aus, die seinem Streben nach Sicherheit entgegen kommen.
4. Man kann nicht entscheiden, ob das Jahr 2000-Problem deshalb keine Folgen hatte, weil es gar kein Problem war, oder nur weil man mit immensem Aufwand so gut vorgesorgt hatte.

9. Fehler werden zur Arbeitsplatzsicherung eingebaut

Zum Glück scheint es mittlerweile seltener geworden zu sein. Früher war es jedoch sehr üblich, dass Entwickler ihre Software möglichst kompliziert gemacht haben, um sich unentbehrlich zu machen. Teilweise sind auch Fehler eingebaut worden, um dem Arbeitgeber im Falle einer Kündigung noch „ein Ei ins Nest zu legen“. Man spekulierte darauf, dass man den Arbeitgeber damit indirekt erpressen könnte: Er würde nach einer Kündigung bald wieder anrufen und darum bitten, dass der Entwickler wieder für das Unternehmen tätig würde, da es sein Nachfolger nicht geschafft habe. Es gibt einige Fälle, in denen der Arbeitgeber dann einen „Sonderpreis“ zahlen durfte, um den alten Entwickler wieder für sich arbeiten zu lassen.

10. Zusammenfassung

Es gibt eine Reihe von allgemeinen Tatsachen, durch die Software und ihre Entwicklung beeinflusst wird. Software wird von kommerziellen und nicht-kommerziellen Urhebern entwickelt. Die Urheber sind häufig abhängig von Zulieferern. Entwickler sind kreative Menschen. Ein intensiver Kontakt zu den Nutzern ist förderlich. Dienstleister leben von schlechter Software, sogar Schad-Software generiert Umsätze. Teilweise werden sogar absichtlich Fehler oder Schwachstellen eingebaut.

IV. Open-Source-Markt im Besonderen

B 1. Existierende Software wird Open-Source

1. weil der Urheber keine kommerziellen Interessen hat

Der Mangel an kommerziellem Interesse kann ein Grund sein, eine bereits existierende Software unter eine Open-Source-Lizenz zu stellen. Beispielsweise kann es sich um Nebenprodukte der Arbeit handeln[1], oder Prototypen, die nicht zu einem Geschäftsabschluss führten. Auch Software, die nicht mehr auf regulärem Weg wirtschaftlich vertrieben werden kann[2], aber eine ausreichende Benutzerplattform hat[3], kann so zur Verfügung gestellt werden. Viele Beiträge stammen von Universitätsangehörigen und dürften im Rahmen ihrer Arbeit entstanden sein[4]. Relativ selten dürfte der Fall sein, dass es den ursprünglichen Auftraggeber nicht stört, da die Software zum internen Gebrauch gedacht ist[5].

1. So wie „The Gimp“ ursprünglich aus einem LISP-Projekt an der Universität entstand, vgl. http://www.acm.org/crossroads/xrds3-4/gimp.html.
2. Beispielsweise weil die Produktlinie abgelöst wurde, die Verkaufszahlen oder die Umsätze zu gering sind.
3. Der Ruf eines Unternehmens kann ernsthaften Schaden nehmen, wenn eines seiner Produkte aus dem Markt genommen wird, aber keine ausreichenden Migrationsmöglichkeiten bestehen.

2. weil der Urheber andere kommerzielle Interessen hat

Manche Software ist so gut wie unverkäuflich. Vielleicht müsste sie zu teuer lizensiert werden[1], oder man möchte einen anderen Schwerpunkt der Geschäftsfelder setzen[2]. In diesen Fällen kann es sinnvoll sein, die Software unter einer Open-Source-Lizenz zu verbreiten. Dadurch gewinnt sie Verbreitung, der Hersteller gewinnt an Bekanntheit und kann Hardware und Dienstleistungen passend zur Software anbieten[3]. Letztlich ist dies eine Variante der „Kuchenvergrösserung“[4].

3. weil die Lizenz ein Alleinstellungsmerkmal ist

Wenn ein Softwarehersteller mit seinem Produkt anderen Anbietern deutlich unterlegen ist, so kann er seine Software unter eine Open-Source-Lizenz stellen. Auf diese Weise hat er ein Alleinstellungsmerkmal und kann relativ sicher sein, dass seine Software bekannt wird. Die Unterlegenheit ist häufig nicht technisch bedingt, sondern liegt in der Unerfahrenheit des Anbieters oder fehlenden Mitteln zu entsprechendem Marketing. Aufgrund der Tendenz des Softwaremarktes zur Benutzung von Marktführern kann man bei Standardsoftware davon ausgehen, dass maximal fünf bis zehn Anbieter eine grosse Verbreitung finden[5] und etwa 80 % des Marktes abdecken. Die anderen Anbieter können bestehen, wenn der Markt insgesamt gross genug ist oder müssen sich auf lukrative Nischen spezialisieren[6]. Ein Anbieter kann sich von einer abschlagenen Wettbewerbsposition auf einen der vordersten Plätze katapultieren, indem er sein Produkt als Open-Source verbreitet[7]. Sobald es dann entsprechende Verbreitung gefunden hat, kann er Hardware und Services anbieten oder ein Parallelprodukt herausbringen, welches nur

4. Vgl. http://orbiten.org/ofss/codd-render.cgi?action=project. Da nur teilweise die Mailadressen genannt sind, fällt eine Zuordnung schwer. Die von CODD vorgenommene Bewertung ist mit Vorsicht zu betrachten, da z.B. jemand durch blosses Umformatieren des Quelltextes an die Spitze der Statistik gelangen kann. Wie schon erwähnt ist Lines of Code (LOC) ist eine äusserst problematische Metrik.
5. Vgl. Raymond S. 128. Sicher kann man argumentieren, die Aufgabe sei ja für den Auftraggeber gelöst. Unfreiwillig unterstützt er dadurch jedoch ggf. seine Mitbewerber, die Aufwendungen bei der Entwicklung einer eigenen Software sparen. Sinnvoller scheint ggf. ein Vertrieb der Software unter Beteiligung des Auftraggebers oder eine gemeinsame Entwicklung der Wettbewerber zu sein.
1. Wenn voraussichtlich nur eine kleine Nutzergruppe zahlen, diese aber hohe Ansprüche stellen würde, so dass man z.B. Support bieten müsste.
2. Z.B. weil die Kernkompetenzen im Hardware-/Dienstleistungsbereich liegen.
3. Vgl. die Strategien bei Raymond S. 134-140.
4. Vgl. zum Begriff: Haft, S. 107 und 168. Durch Erweiterung des Gesamtmarktes werden selbst geringe prozentuale Marktanteile interessant.
5. Beispiel EJB-Container: Bea, IBM, Oracle, Sun decken den grössten Marktanteil ab. Datenbanken: IBM, Oracle, Microsoft, Sybase, Software AG. Office-Pakete: Microsoft, Sun/StarDivision, IBM/Lotus.
6. Beispiel: ChemOffice mit einem Office-Paket für Chemiker.
7. So z.B. JBoss bei EJB-Containern, MySQL bei Datenbanken oder StarOffice bei Office-Paketen.

gegen reguläre Lizenz erhältlich ist[1].

4. Zusammenfassung

Die Veröffentlichung als Open-Source wird genutzt, wenn der Urheber kein direktes kommerzielles Interesse (mehr) an der Software hat. Sie taugt auch als Alleinstellungsmerkmal.

B 2. Open-Source wird neu entwickelt

1. aus Spass

Einige Autoren geben an, die Software einfach aus Spass entwickelt zu haben. Das Betriebssystem Linux ist anfangs so entstanden. Die Entwickler waren als Schüler, Studenten oder Arbeitnehmer tätig und haben in ihrer Freizeit an der Software gearbeitet. Sie verfolgten keine kommerziellen Interessen und fanden es gut, in einem netten Team mit anderen engagierten Mitstreitern etwas zu erschaffen.

2. um etwas zu beweisen

Manche Softwareentwickler reizt es zu zeigen, dass eine bestimmte Software besser zu implementieren oder günstiger zu vertreiben wäre. Ein Beispiel dafür mag JBoss sein. Zwar gibt es viele EJB-Container. Diese sind jedoch teuer[2] und können nicht vom Nutzer erweitert werden. Viele würden gerne einen EJB-Container nutzen, doch der Preis hält sie davon ab und verhindert eine weite Verbreitung. Die Entwickler treten deshalb an, um zu zeigen, dass ein solches Programm auch mit „Hausmitteln“ erstellt und umsonst verbreitet werden kann. Die Entwicklungskosten bekommen sie z.B. durch Schulungen oder den Verkauf von Dokumentation herein.

3. um bekannt zu werden

Software kann als „Gesellenstück“ erstellt und verbreitet werden, um die Kompetenz des Autors zu beweisen und ihn bekannt zu machen. Durch ein nützliches Programm, das kostenlos ist und rasant Verbreitung findet, kann der Autor sich weltweit an die Spitze katapultieren. Linus Torvalds oder die Entwickler von „The Gimp“ sind ein gutes Beispiel dafür. McGowan merkt kritisch an, dass es drei Bereiche gibt, in denen man eine Reputation erwerben kann: Technik, Management oder Teamarbeit[3]. Dabei erfordert Technik die Lösung schwieriger Probleme, die ggf. sozial wenig wertvoll sind. Im Bereich Management wäre plausibel, dass ggf. unnötigerweise neue Projekte gestartet werden, da-

1. Sun Microsystems hat diese Wendung bei StarOffice vollzogen.
2. So kostet z.B. BEA WebLogic pro Prozessor 15.460 bzw. 26.290 Euro.
3. Vgl. McGowan 275.

mit man deren Verwalter sein kann. Bei Teamarbeit ist das Projektziel gleichgültig, so lange das Team gross genug ist. Alles in allem ergeben sich daraus keine gesamtwirtschaftlich interessanten Anreize, insbesondere nicht, da für den Grossteil geeigneter Kandidaten herkömmliche Firmen (ggf. ihre eigenen) ebenso grosse Chancen bieten dürften.

4. um die Basar-Methode zu nutzen

Einzelentwickler oder kleine Unternehmen haben nicht ausreichende organisatorische und finanzielle Möglichkeiten, um eine Software in grossem Stil zu entwickeln. Problematisch ist beispielsweise die Anforderungsanalyse[1] oder die Bezahlung von Testern[2]. Gerade in diesen Punkten hilft die Basar-Methode[3]: Die Software wird inkrementell zusammen mit den Benutzern erstellt[4]. Sie diskutieren mit den Entwicklern über die Anforderungen, machen Verbesserungsvorschläge und melden Fehler. Dies geschieht kostenlos, während eine „Cathedral"-Firma[5] dafür viel Geld ausgeben muss[6].

5. wenn ein existierendes Produkt nicht verfügbar ist

Es mag zwar Produkte geben, die den Anforderungen genügen, doch diese sind oft nicht verfügbar. Das kann am Preis oder den Lizenzbedingungen liegen. Dann dient das existierende Produkt als Vorlage für eine Open-Source-Entwicklung[7].

6. wenn ein existierendes Produkt Mängel hat

Für jeden Nutzer ist es ärgerlich, wenn ein Programm fehlerhaft ist. Softwareentwickler kontaktieren zunächst den Hersteller. Wenn dieser keine Abhilfe schafft, fragen sie sich zunächst, ob sie das Programm korrigieren können. Wenn das nicht möglich ist[8], entsteht oft der Anreiz, ein neues, besseres Programm zu entwickeln. Aber auch wenn einer Software Fähigkeiten fehlen, die der Anbieter hartnäckig nicht einbaut, entsteht ein solcher Anreiz.

7. als Alleinstellungsmerkmal

Kommerzielle Hersteller von Software erstellen Open-Source-Software

1. Um eine erfolgreiche Software auf den Markt zu bringen, können Studien notwendig sein, welche Zielgruppe in Frage kommt und welche Fähigkeiten die Software haben muss, um akzeptiert zu werden.
2. Um eine Software auf ein qualitativ hochwertiges Niveau zu bringen, sind u.a. viele Tests notwendig.
3. Vgl. Raymond S. 21 f.
4. Vgl. Raymond S. 27, er beschreibt die Benutzer als „Co-Developers". Es ist zweifelhaft, ob dies ein effizienter Ansatz ist.
5. Vgl. Raymond S. 21.
6. Vgl. Raymond S. 38.
7. Ein Beispiel ist „The Gimp" mit der Ähnlichkeit zu Adobe Photoshop, JBoss als „WebLogic killer" oder Linux als Ersatz-Unix.
8. Kein Quelltext oder Dokumentation vorhanden, zu kompliziert.

neu, obwohl es entsprechende Programme schon unter anderen Lizenzen gibt, z.T. sogar in grosser Zahl[1]. Der Grund liegt darin, dass sie ein Alleinstellungsmerkmal in der Lizenz sehen. Auch die Distributoren von Open-Source gehören in diese Gruppe.

8. Zusammenfassung

Open-Source wird aus Spass neu entwickelt, um etwas zu beweisen, bekannt zu werden, die Basar-Methode zu nutzen oder weil existierende Produkte nicht verfügbar sind oder Mängel haben. Kommerzielle Urheber sehen in ihr allerdings eher ein Alleinstellungsmerkmal.

B 3. Open-Source wird nicht neu entwickelt

1. wenn die Entwickler kein Eigeninteresse haben

Die Entwicklung von Open-Source-Software hängt in hohem Masse vom Eigeninteresse der Entwickler ab. Nur wenn sie selbst ein uneigennütziges Interesse haben oder damit rechnen, über Dienstleistungen oder Hardware die Kosten der Entwicklung hereinzubekommen, entwickeln sie Open-Source. Daher ist es schwer vorstellbar, dass Open-Source sich in Bereiche ausdehnt, in denen schon ein reiches Angebot besteht, die existierenden Produkte zu einigermassen akzeptablen Bedingungen angeboten werden, der Erstellungsaufwand hoch ist, kein technischer Reiz vorhanden ist[2] und die Open-Source-Lizenz kein Alleinstellungsmerkmal mehr ist[3].

2. wenn die Basar-Methode scheitert

Die Basar-Methode kann scheitern[4]. Einmal kann es daran liegen, dass die Nutzer keine ausreichende fachliche Kompetenz besitzen, um ihre Anforderungen zu beschreiben. Zwar sind theoretisch methodenbedingt mehr Nutzer in der Lage, ihre Meinung zu äussern[5]. Damit die Basar-Methode funktioniert, müssen jedoch die richtigen Nutzer zur richtigen Zeit mitmachen[6]. Ausserdem kann gerade durch die Vielzahl von Stimmen das Problem entstehen, die richtige Richtung auszufiltern[7]. Man

1. z.B. MySQL, vgl. http://www.mysql.com.
2. Beispielsweise: CAD-Applikationen, BWL-Anwendungen, aufwendige Spiele.
3. Entwicklung und Pflege weiterer Datenbanksoftware neben MySQL.
4. Man sehe sich dazu die „Projektleichen“ der Open-Source-Portale http://www.sourceforge.net oder http://www.berlios.de an.
5. Weil Hierarchien keine Rolle spielen. In herkömmlichen Projekten kann z.B. der Sachbearbeiter nicht mit über die Anforderungen diskutieren, weil sein Vorgesetzter dieses Privileg ggf. nutzt, um seine Position zu festigen. Bei der Basar-Methode können alle Sachbearbeiter ebenso mitdiskutieren wie der Behördenleiter, die Reinigungsfachkräfte oder externe Schüler und Studenten.
6. Es bringt nichts, wenn alle Nutzer sich darüber einig sind, dass die Software zu wenig Funktionalität hat, wenn sie nicht in der Lage sind, die Anforderungen ausreichend (ggf. im Dialog mit den Entwicklern) zu präzisieren.
7. Die Diskussionsrichtungen müssen entsprechend ausgewertet werden.

stelle sich vor, mehrere divergierende Linien enstehen: Ein Teil der Nutzer möchte „linksherum“, der andere „rechtsherum“. Zunächst ist das Problem, dies überhaupt festzustellen. Dazu muss man die Kommunikation erst einmal auswerten. Hat man dies getan und steht fest, wie genau die Anforderungen aussehen, müssen sich die Entwickler entscheiden. Sie können die Anforderungen beide aufnehmen (sofern sie sich nicht ausschliessen) und nacheinander abarbeiten. Dann müssen die der unterlegenen Richtung zugewandten Entwickler bereit sein, auch die andere Entscheidung mitzutragen und ihre Arbeitszeit dafür zu investieren. Oder sie weisen eine Richtung gemeinsam ab, wodurch sich ein Teil der Nutzer[1] aus dem Projekt verabschiedet. Schliesslich gibt es noch die Möglichkeit der Teilung des Gesamtprojektes, so dass fortan zwei Versionen des Projektes existieren, die sich von Tag zu Tag mehr unterscheiden werden[2]. Daran ist kritisch, dass sich auch die Entwickler aufteilen. Üblicherweise gibt es deutlich weniger Entwickler als Nutzer. Derartige Aufteilungen der Entwicklerkapazitäten können schnell das Projekt insgesamt zum Erliegen bringen. Die Aufsplittung bedeutet auch, dass sich die potentielle Nutzergruppe reduziert. Die ist aber für die Refinanzierung über Hardwareverkauf und Dienstleistungen wichtig.

Auch die technische Kompetenz der Nutzer ist wichtig. Schliesslich lebt die Basar-Methode davon, dass aus Nutzern irgendwann Mitentwickler werden. Je weiter die typischen Nutzer jedoch von der Technik entfernt sind, desto unwahrscheinlicher wird ihre Beteiligung[3].

Die Basar-Methode hat auch Probleme, wenn es um Skalierbarkeit geht. Es ist bekannt, dass Teams ineffizient werden, sobald sie aus deutlich mehr als 10 Personen bestehen. In diesem Fall steigt der Koordinationsaufwand. Hinzu kommt, dass die Stillstandzeiten zunehmen[4]. Der Erfolg der Basar-Methode hängt also stark mit der Partitionierbarkeit der Aufgabe zusammen. Selbst wenn diese gegeben ist, heisst das aber noch nicht, dass die Entwickler diese auch tatsächlich erkennen und nutzen können. Denn die Teambildung hängt auch stark von persönlichen

1. Und damit auch ein Teil der billigen Tester.
2. Weil man an beiden weiter arbeitet. Es ist ein enormer Aufwand, später diese Versionen wieder zusammen zu führen („mergen“).
3. Ein typischer Buchhalter wird vermutlich wenig Interesse haben, sich an einer Softwareentwicklung zu beteiligen.
4. Das liegt an der beschränkten Parallelisierbarkeit von Aufgaben. Es ist häufig der Fall, dass alle Entwickler eine bestimmte Resource (z.B. Datei) benötigen. Während ein Entwickler sie benutzt, müssen die anderen darauf warten.

Sympathien ab[1]. Damit kämpfen viele Softwareprojekte. Ferner sind Glaubensfragen schon immer kritisch gewesen[2].

Ein wichtiger Einwand kommt quasi vom Erfinder der „Cathedral"-Methode[3]. Brooks meint, dass die meisten Programme viel schlimmere konzeptionelle Brüche aufweisen als die meisten Kathedralen[4]. Dabei sprach er von herkömmlichen Softwareprojekten. Erst recht ist dies bei Open-Source-Projekten anzunehmen, sofern der Basar-Stil tatsächlich gelebt wird. Raymond hält den Basar-Stil daher in der Startphase eines Projektes nicht für sinnvoll[5].

3. wenn rechtliche Probleme befürchtet werden

Die meisten Entwickler von Open-Source haben keinerlei Interesse, Software zu entwickeln, wenn rechtliche Probleme zu befürchten sind[6]. Dabei ist es nicht wichtig, ob sich diese bei Prüfung der Rechtslage erhärten. Es geht primär um das subjektive Empfinden, das von der Überzeugung genährt wird, dass Rechtsstreitigkeiten von typischen Softwareentwicklern nicht eingeschätzt werden können und höchstens den beteiligten Juristen zu Reichtum verhelfen. Daher behindert eine unübersichtliche oder instabile Rechtslage die Entwicklung von Open-Source-Software. Ein Hauptanliegen vieler Entwickler ist, Software-Patente zu verhindern[7]. Man befürchtet, diese würden jede Kreativität im Keim ersticken, weil man ständig Patentrecherchen durchführen müsse und selbst dann noch keine endgültige Sicherheit erlangt hätte[8].

4. wenn es zu viel Vorwissen erfordert

Die sinnvolle Beteiligung an einem Projekt kann ein bestimmtes fachliches oder technisches Vorwissen erfordern. So dürfte es schwer sein, z.B. eine anspruchsvolle Chemie- oder Sozialamts-Software[9] zu ent-

1. Vgl. Williams S. 157 zum Verhältnis von Stallman, Raymond und Torvalds.
2. Vgl. Williams S. 166 zum Angriff von Raymond auf den Urheber von Tcl/Tk.
3. Vgl. Brooks S. 42, der zur Begründung der konzeptionellen Integrität eine Kathedrale betrachtet, die von acht Generationen gebaut wurde. Anstatt jeweils ihren eigenen Stil einzubringen schufen die Baumeister ein Werk von hoher Reinheit.
4. Weil zwar viele gute Ideen eingebracht werden, diese aber nicht zum Konzept passen oder es bestenfalls verwässern.
5. Vgl. Raymond S. 47, nur zum Testen, Fehlersuchen und Verbessern.
6. Denn das ist gerade ein Grund für Open-Source, vgl. Raymond S. 133.
7. Vgl. http://swpat.ffii.org/index.de.html.
8. Ein Gegenargument ist, dass im Maschinenbau, der Elektrotechnik und Chemie längst Patente gelten und es dort dennoch kleine und mittelständische Unternehmen gibt. Dort existiert aber eine andere Arbeitsweise als in der Informatik.
9. Vgl. zu den Schwierigkeiten kommerzieller Anbieter: http://www.computerwoche.de/index.cfm?pageid=267&type=ArtikelDetail&id=143244&cfid=5232291&cftoken=6619834&nr=1 („Oracle setzt Berliner Großprojekt in den Sand", COMPUTERWOCHE Nr. 34 vom 25.08.2000) oder http://www.computerwoche.de/index.cfm?pageid=267&type=ArtikelDetail&id=135620&cfid=5232291&cftoken=6619834&nr=3 („Überalterte Software stürzt Sozialämter ins Chaos", COMPUTERWOCHE Nr. 36 vom 10.09.1999).

wickeln, wenn man keinerlei Vorbildung in diesem Bereich hat. Natürlich kann man sich im Laufe des Projektes manches Wissen aneignen. Jedoch müssen die Entwickler zusammen mit den Fachexperten eine gemeinsame Kommunikationsbasis finden. Wenn das Vorwissen zu unterschiedlich ist, wird diese nicht zustande kommen, zumal die Beteiligten sehr freiwillig teilnehmen und der Abstand zu ggf. bereits existierenden Closed-Source-Produkten unerreichbar scheint[1]. Auch im technischen Bereich kann es zu Problemen kommen, wenn die geplante Lösung technisch sehr anspruchsvoll ist, aber nur unerfahrene Entwickler verfügbar sind.

5. Zusammenfassung

Open-Source wird nicht entwickelt, wenn die Entwickler kein Eigeninteresse haben, die Basar-Methode scheitert, rechtliche Probleme befürchtet werden oder es zu viel Vorwissen erfordert.

B 4. Open-Source wird nur distributiert

1. wenn eine grosse Anzahl Nutzer erwartet wird

Distributoren für Open-Source finden sich nur, wenn eine grosse[2] Anzahl von Benutzern zu erwarten ist. Grundsätzlich muss es sich um eine Standardsoftware in dem Sinne handeln, dass sie mit geringen oder gar keinen Änderungen von einer grossen Zielgruppe benutzt werden kann. Das liegt daran, dass die Distributoren sich auf lukrative Programme konzentrieren müssen. Schliesslich wollen sie mit dem passenden Knowhow Geld verdienen.

2. wenn mit Support Geld zu verdienen ist

Mit dem Support des Programmes muss Geld zu verdienen sein. Damit scheiden Programme aus, die sehr einfach oder selbsterklärend sind. Linux ist ein gutes Beispiel für eine support-trächtige Software: Unix-Systeme gelten generell als kompliziert und konfigurationsbedürftig[3]. Man kann Schulungen für Administratoren und Benutzer anbieten, Installationen im Kundenauftrag durchführen und Beratertage verkaufen[4]. Auch mit pflegebedürftiger Software lässt sich Geld verdienen[5].

1. Man stelle sich einen Fachexperten vor, der dem Entwickler erst ein Jahr lang in seiner Freizeit die Grundlagen vermitteln muss, bevor – wenn überhaupt – erste Ergebnisse sichtbar werden. Während dessen könnte er trotz Lizenzkosten existierende Closed-Source-Software nutzen.
2. Der Begriff „gross“ ist zu verstehen als „ausreichend gross“. Eben so, dass man sich von den Umsätzen zumindest ernähren kann.
3. Ausnahmen dürften NeXTStep, vgl. http://www120.pair.com/mccarthy/nextstep/intro.htmld bzw. Mac OS X, vgl. http://www.apple.com/de/macosx, sein.

3. wenn es kostenpflichtige Zusatzprodukte gibt

Der Distributor kann auch mitverdienen, indem er Nicht-Open-Source-Software zusammen mit der Open-Source-Software anbietet. Ein Linux-Distributor kann darauf basierende Datenbank-, Mail- oder Firewall-Systeme anbieten[1]. Auch Officeprodukte, Entwicklertools oder Spiele gehören dazu[2]. Der Distributor kann auch kostenpflichtige Abkömmlinge der Open-Source-Software vertreiben, die selbst nicht als Open-Source vertrieben werden[3]. Auch Bücher sind sehr beliebte Zusatzprodukte[4], zumal die Entwickler der Software als Autoren prädestiniert sind[5]. Die GNU Free Documentation License fristet hingegen ein Nischendasein[6].

4. wenn es den Hardwareabsatz fördert

Die Auswahl an möglicher Hardware ist sehr gross. Bei einem Betriebssystem kommt es z.B. darauf an, dass entsprechende Gerätetreiber für die Hardware verfügbar sind. Sonst funktionieren die Geräte nicht oder nicht optimal. Aber auch bei Datenbanksoftware kann es sinnvoll sein, spezielle Hardware einzusetzen[7]. Viele Kunden gehen davon aus, dass der Distributor der Open-Source-Software entsprechende Erfahrung in der Auswahl der Hardware besitzt[8]. Auf diese Weise kann er daran mitverdienen. Aber auch Hardwarehersteller können angepasste Open-Source-Software vertreiben[9]. Dann können sie dem Kunden, der die Open-Source-Software nutzen möchte, völlig neue Perspektiven eröffnen[10] und sich selbst neue Absatzfelder sichern.

4. Vgl. http://www.suse.de/de/services/index.html; die Palette reicht von 0190-Telefonsupport zu 1,86 Euro/Min., über Support-Calls für 46,60 Euro pro Fall oder Emergency-Callpack mit max. 2 Stunden Reaktionszeit für 600 Euro pro Fall bis hin zur Individualprogrammierung nach Aufwandsschätzung.
5. JBoss bietet als Hersteller (nicht Distributor) für $ 10.000 pro Jahr Support, oder für $175 auf Stundenbasis; vgl. http://www.jboss.org/jbossgroup/services.jsp.
1. Vgl. http://www.suse.de/de/products/index.html.
2. Z.B. http://www.suse.de/de/products/software/index.html.
3. Vgl. ArgoUML (OS) und Poseidon (Developer Edition für $ 399), http://www.argouml.com, http://www.gentleware.com.
4. Vgl. SuSE Press, http://www.suse.de/de/products/books/index.html.
5. Ein Verlag, der aus seiner Unix-Historie heraus schon früh Bücher zu Open-Source-Themen veröffentlicht hat, war O'Reilly, http://www.oreilly.com. Stallman war der Meinung, auch die Dokumentation müsse frei sein, während Tim O'Reilly dies (interessengemäss) strikt ablehnte, vgl. Williams S. 163.
6. Vgl. die abgedruckte Version bei Williams S. 209-216.
7. Z.B. RAID-Systeme gegen Datenverlust und zur Performancesteigerung.
8. Früher hat auch SuSE Hardware angeboten. Zur Zeit kann man unter http://www.lisa.de ein Beispiel für den Hardwarevertrieb mit Open-Source finden.
9. Beispielsweise IBM.
10. IBM bietet einen „Linux-Mainframe" ab $ 400.000 an, http://www.heise.de/newsticker/data/odi-25.01.02-001, der deutlich stabiler ausgelegt ist als herkömmliche PC-Hardware. Auf einem solchen Rechner lassen sich mehrere tausend Linux-Installationen parallel betreiben, http://www.heise.de/newsticker/data/odi-07.12.00-001.

5. Zusammenfassung

Die Chancen für eine Distribution von Open-Source-Software steigen, wenn eine grosse Anzahl von Nutzern erwartet wird und vermutlich mit Support, Hardware oder Zusatzprodukten Geld zu verdienen ist.

V. Zusammenfassung

Für die Entwicklung von Software und insbesondere Open-Source gibt es eine grosse Anzahl von Anreizen und Hemmnissen. Der vom Urheberrecht ausgehende Anreiz ist nur ein Aspekt unter vielen. Er hat durch die mangelhafte Implementierung des Urheberrechts nur eine nachrangige Bedeutung.

C. Widerspruch zu Grundannahmen des Urheberrechts

I. Widerspruch

Es könnte sein, dass die fortlaufende Erstellung von Software unter Open-Source-Lizenzen ein Widerspruch zu den genannten Grundannahmen des Urheberrechts darstellt.

1. Kostenlose Nutzungserlaubnis

Die Entwickler geben ihre Werke ganz oder weitestgehend kostenlos ab. Das Urheberrecht wurde in der historischen Entwicklung geschaffen, weil man annahm, die Urheber würden es zur Refinanzierung der Werkschöpfung benötigen. Das ist nun bei Open-Source-Software auf den ersten Blick nicht der Fall[1].

Aber auch ausserhalb von Open-Source gibt es Urheber, die ihre Werke kostenlos verbreiten. Man denke z.B. an Schöpfungen, mit denen gar keine Gewinnerzielungsabsicht verfolgt wird[2]. Diese Werke gab es schon lange vor Einführung des Urheberrechts im heutigen Sinne. Man muss daher unterscheiden: Das Urheberrecht möchte eine Möglichkeit zur Verfügung stellen, *auch* eine kostenpflichtige und nicht neutrale Lizensierung zu ermöglichen. Es hält aber keinen Urheber davon ab, seine Werke zu verschenken, wenn er glaubt, sich dies leisten zu können. Das Urheberrecht hatte nie die Zielsetzung, die kostenlose Einräumung von Nutzungsrechten zu regulieren. Wenn die Open-Source-Urheber ihre Werke kostenlos abgeben wollen, so hat das Urheberrecht nichts dagegen. Denn das hindert andere Urheber nicht daran, für die Nutzungs-

1. Die Open-Source-Schöpfer nutzen das Urheberrecht natürlich, um die Open-Source-Lizenzen durchzusetzen.
2. Z.B. zu Werbezwecken erstellte Werke oder einfach eine Vereinszeitung.

rechte an ihren Werken Geld zu verlangen[1]. In dieser Hinsicht verhält sich das Urheberrecht neutral, es bestraft den kostenlosen Nutzungsrechtseinräumer ebensowenig wie es ihn fördert.

2. Freie Verbreitung

Die Entwickler nutzen das Urheberrecht dazu, die freie Verbreitung ihrer Werke zu sichern. Sie wollen die Verbreitung gar nicht einschränken. Im Gegenteil, sie bezwecken, dass ihre Werke möglichst oft kopiert und verwendet werden, da ihre Geschäftsmodelle auf einer weiten Verbreitung der Software basieren[2]. Zu diesem Zwecke haben die weit verbreiteten Open-Source-Lizenzen entsprechende Formulierungen[3]. Die Lizenzen unterscheiden sich jedoch in einigen Punkten.

a) Rückgabe der Bearbeitungen unter gleicher Lizenz

Einige Lizenzen[4] fordern, dass Bearbeitungen (z.B. Fehlerkorrekturen) wieder unter der gleichen Lizenz verbreitet werden müssen. Der Urheber möchte dadurch sicherstellen, dass sich nicht nur die ggf. fehlerbehaftete Originalversion verbreitet, sondern ebenso die Korrekturen.

b) Kommerzielle Nutzung erlaubt

Die kommerzielle Nutzung der Software ist bei manchen Lizenzen verboten. Dahinter steht die Überlegung, dass kommerzielle Nutzer die Software auch kostenpflichtig lizensieren sollen, nicht kommerzielle Nutzer sie hingegen frei nutzen können sollen.

c) Kommerzieller Vertrieb erlaubt

Einige Lizenzen fordern, dass die Verbreitung nur kostenlos erfolgen darf. Grundsätzlich ist Open-Source nicht frei im Sinne von „free beer“, sondern nur im Sinne von „free speech“[5]. Das bedeutet, dass ein Anbieter von Open-Source-Software durchaus tausende von Euro für die Software kassieren kann, aber er muss eben den Quelltext zur Verfügung stellen. Grund eines Verbots des kommerziellen Vertriebs kann sein, dass man verhindern möchte, dass der Benutzer übervorteilt wird[6].

1. Natürlich kann dies den Markt verzerren. Wenn ein Urheber ein Werk kostenlos abgibt und ein anderer für ein vergleichbares Werk einen enormen Preis erzielen möchte, kann das aber der Markt regeln (oder das Wettbewerbs-/Kartellrecht). Das zwingt den Urheber mit dem höheren Preis dazu, diesen zu begründen.
2. Support und Hardwareverkauf werden erst bei einer breiten Installationsbasis lukrativ. Bei zahlreichen nicht-kommerziellen Programmen freut sich der Urheber auch über eine weite Verbreitung, da sie seine Bekanntheit erhöht.
3. Vgl. die GPL Preamble bei Jaeger/Metzger S. 177.
4. Z.B. GPL und LGPL, vgl. DiBona/Ockham/Stone/Perens S. 185.
5. Vgl. Raymond S. 176.
6. Berühmt geworden ist der dreiste „Verkauf“ des kostenlos verfügbaren Unix-Editors vi für über 10.000 Euro. Der Kunde wird sich geärgert haben, denn exakt die gleiche Software hätte er kostenlos aus dem Internet haben können. Andererseits war ihm der Editor offenbar auch so viel wert und wurde dringend benötigt. Letztlich hat er den Verkäufer für das Knowhow des Findens im Internet entlohnt.

Zudem könnte der Benutzer den Eindruck erhalten, er habe teuer für die Software bezahlt und daher auch Ansprüche[1] gegen den Urheber. Dies könnte auf den Ruf des Urhebers zurückschlagen[2].

d) Inkludieren/Masquerading erlaubt

Möglich wäre auch, Open-Source-Software zu nehmen, sie minimal zu erweitern und diese dann kompiliert als selbsterstelltes Produkt ohne Nennung der Open-Source-Quellen zu vertreiben. Das ermöglicht die kuriose Situation, dass ein Unternehmen sich öffentlich gegen Open-Source stellt, in seine Produkte aber gerne fremde Sourcen aufnimmt, so lange es nutzt[3]. Dies betrachten einige OS-Urheber als unfair und schliessen es in den Lizenzbedingungen aus.

e) Zusammenfassung

Das Urheberrecht hilft Open-Source-Urhebern durch Property Rights, bestimmte Erscheinungsformen (z.B. kommerzielle Nutzung oder Verunstaltungen) zu verhindern. Würden diese nicht existieren, könnte die Erstellung von Open-Source darunter leiden[4]. Andererseits gelten aber auch hier die (später ausführlich zu besprechenden) Probleme der Aktivlegitimation der Urheber[5].

3. Anreiz zur Werkschöpfung

Die Urheber haben viele Anreize, Open-Source zu entwickeln. Die meisten Anreize haben überhaupt keine oder nur eine äusserst schwache Beziehung zum Urheberrecht. Würde man das Urheberrecht für Software abschaffen, so würde immer noch Software entwickelt[6]. Das Urheberrecht ist zur Zeit äusserst mangelhaft implementiert, und zwar quer durch alle gesellschaftlichen Schichten und alle Branchen. Es ist

1. Nicht unbedingt im rechtlichen, eher im moralischen Sinne.
2. Dieser hat die Software „verschenkt“, der Kunde einen hohen Betrag an den Distributor bezahlt. Nun ist der Kunde ggf. verärgert, weil die Software Fehler hat oder vom Funktionsumfang her nicht ausreicht. Dieser Ärger könnte sich leicht gegen den Hersteller, statt den Distributor, richten, obwohl der Hersteller den Distributor vielleicht nicht einmal kennt.
3. So basiert das Programm ftp.exe von Microsoft Windows 2000 auf offenen Sourcen der Universität Berkeley. In der Datei findet sich jedenfalls ab der Byteposition hexadezimal 5415 der Text „Copyright (c) 1983 The Regents of the University of California. All rights reserved.“, ähnlich bei finger.exe, nslookup.exe, rcp.exe und rsh.exe (mit anderen Jahreszahlen).
4. Z.B. weil Urheber keine Werke erstellten, falls diese später zwar verwendet, aber vom Verwender verleugnet würden.
5. Kurz: Es ist fraglich, ob z.B. Linus Torvalds seine Rechte am Linux-Kernel durchsetzen könnte, da er dazu gemeinsam mit allen, ggf. sogar unbekannten und über die ganze Welt verstreuten, Miturhebern gegen den Verletzer klagen müsste. Unter diesem Aspekt erscheint letztlich die gesamte Lizenzvergabe überflüssig. Es könnte sein, dass sich die Nutzer lediglich aus juristischer Unwissenheit um dieses Klägerproblem so nett verhalten und die Lizenz befolgen.
6. Man denke an die gesamte Individualsoftware: Diese kann auch ohne Urheberrecht gut geheim gehalten werden (zumindest im Quelltext) und ist oft derart spezialisiert, dass sie nicht einmal ein Mitbewerber nutzen könnte, weil sie einfach nicht zu seinem Betrieb passt.

kaum im Volk verankert: Man vergleiche die Reaktion auf einen Autodiebstahl und einen „geistigen Diebstahl“[1]. Es könnte sogar sein, dass die Branche im Laufe der historischen Entwicklung andere Mechanismen gefunden hat, um ihre Arbeit zu schützen[2].

4. Refinanzierung der Werkschöpfung

Es wurden zahlreichen Möglichkeiten aufgezeigt, die Werkschöpfung zu refinanzieren. Diese kann man unter Ausblendung vieler wichtiger Aspekte auf Dienstleistungen und Hardware reduzieren. Es ist möglich, dass Open-Source lediglich eine Reaktion der Urheber auf die monopolfördernden Umstände des Marktes ist[3]. Die ausführbaren Dateien sind sowieso nicht wirksam schützbar (Kopierproblem), und die Quelltexte sind ohne eine vollständige Entwicklungsdokumentation und Knowhow wenig wert[4].

Die Refinanzierung hängt von zwei Aspekten ab: Den Erlösen und den Aufwänden. Zwar mögen die Erlöse bei Open-Source geringer sein. Man hat dem gegenüber aber auch weniger oder keine Aufwände z.B. für Marketing oder Qualitätssicherung. Die Refinanzierung gelingt, wenn die Erlöse grösser oder gleich den Aufwänden sind. Bei geringen Aufwänden reichen also auch geringe Erlöse. Ferner steigt das Risiko, dass die Refinanzierung misslingt, je höher die Aufwände sind. Es scheint daher so, als ob die Urheber sich z.B. das Geld für Marketing sparen, das den Refinanzierungsbedarf erhöhen würde, denn die Software würde von den sie interessierenden Nutzern eh im Laufe der Zeit weitergegeben. Man kann Open-Source also als Strategie zur Refinanzierungsbedarfs- und damit Risikominimierung betrachten: Die Urheber wollen zwar keine Entlohnung, übernehmen dafür aber auch keinerlei Verpflichtungen[5]. Das verschafft ihnen eine gute Ausgangsposition

1. Zumindest wenn dieser nur die illegale Nutzung betrifft. Wenn jemand das Werk eines anderen als sein eigenes ausgibt, mag dies anders sein.
2. Es werden nur ausführbare Dateien oder Quelltexte veröffentlicht.
3. Erst versuchte man Software über Dongles etc. zu schützen. Praktisch war das nicht möglich, weil es zu viele Umgehungsmöglichkeiten gab. Dann kam der „Browser-Krieg“ zwischen Netscape und Microsoft, vgl. Hoch/Roeding/Purkert/Lindner S. 57 ff. oder http://www.computerwoche.de/index.cfm?pageid=254&artid=11430, der zeigte, dass man mit Marktmacht andere Firmen vernichten kann, indem man ihre Haupteinnahmequelle durch Verschenken eigener Software zerstört. Die Hersteller können sich fragen, warum man überhaupt viel Geld in herkömmliche Werbung stecken soll und nicht versucht, das Geld gleich mit Dienstleistungen und Hardware zu machen. Letztlich mag Open-Source Microsoft am härtesten treffen, denn sie sind eine typische Retail-Firma, die durch Vervielfältigung von Standardprodukten gross und mächtig geworden ist, aber überhaupt keine Reputation im Beratungsbereich oder gar bei Hardware hat.
4. Vgl. Brooks S. 166, der 1975 schon mehr Dokumentation zum Ändern eines Programmes für notwendig hielt, als die meisten Open-Source-Programme bieten.
5. Auch proprietäre Hersteller wollen solche möglichst nicht übernehmen.

selbst gegen Monopolisten, die ihre Produkte verschenken[1].

Eine dem Verlagswesen vergleichbare Institution, also eine klare Trennung zwischen Produzenten und Vermarktern/Distributoren, gibt es auf dem Softwaremarkt nicht. Die wenigen Ansätze[2] dazu erfordern bereits eine vergleichsweise hohe Qualität und Vertragsabschlüsse mit entsprechenden Ansprüchen, versprechen andererseits aber nur geringe sichere Einnahmen[3]. Dieses Verlagswesen dürfte auch daran scheitern, dass es sich bei Software regelmässig um ein komplexes Produkt handelt, welches sich von einem einzelnen Urheber nur schwer produzieren lässt. Auch bei Verlagen sind Werke von mehreren Urhebern eher die Ausnahme, da dies bereits einen gewissen Organisationsgrad voraussetzt und dieser wiederum eher zur Gründung kleiner Firmen führt.

Die Interessen „gelegentlicher“ oder „kleiner“ Urheber sind also für Software am ehesten im Open-Source-Modell gewahrt. Es dient als Katalysator und setzt das dort vorhandene Potential frei.

5. Zusammenfassung

Die kostenlose Nutzungserlaubnis und freie Verbreitung stehen nicht im Widerspruch zum Urheberrecht. Die Urheber haben zahlreiche Anreize und Möglichkeiten zur Refinanzierung der Werkschöpfung, die vom Urheberrecht völlig unabhängig sind. Das Urheberrecht verhält sich neutral bezüglich der kostenlosen Nutzungseinräumung. Es scheint so, dass die Urheber mit Open-Source lediglich eine neue Strategie gefunden haben, ihre Produkte im Markt zu verbreiten und ihre Refinanzierungschancen angesichts übermächtiger Wettbewerber zu erhöhen. Die bei Open-Source vertretenen Elemente sind durchweg bereits aus der Geschichte bekannt. Sie sind allenfalls in ihrer Kombination für die heutige Zeit neu.

II. Überdenkenswert

Die Grundannahmen könnten in zwei Richtungen modifiziert werden.

1. Verbot kostenloser Nutzungseinräumung

Das Urheberrecht könnte seine neutrale Haltung bezüglich der freien Abgabe von Werken aufgeben und diese untersagen. Ein solcher Vorschlag könnte dadurch motiviert sein, dass das Verschenken von Nutzungslizenzen potentiell Schaden bei anderen Urhebern anrichten

1. Vgl. den „Browser-Krieg“ zwischen Microsoft und Netscape.
2. Vgl. z.B. den schon genannten Vertrieb über Data Becker.
3. Vermutlich liegen diese auch bei etwa 10 % des Verkaufspreises. Data Becker dürfte jedoch seinen Vertriebsschwerpunkt bei günstigen Produkten haben, also Preisen zwischen 15 und eher selten 100 Euro, vgl. http://www.databecker.de.

kann[1]. Ein Verbot scheitert jedoch schon daran, dass sich nicht entscheiden lässt, welcher Preis als „kostenlos“ zu bewerten ist. Wenn jemand eine Software, die er für hunderte oder tausende Euro lizensieren könnte, für 5 Euro abgibt, ist dies zwar nicht kostenlos, kann aber die gleiche Wirkung erzielen. Derartige Regelungen wären im Urheberrecht falsch aufgehoben und gehören ins Wettbewerbsrecht.

2. Förderung kostenloser Nutzungseinräumung

Die kostenlose Nutzungseinräumung könnte durch das Urheberrecht gefördert werden. So könnte z.B. dem Urheber eines Werkes, das kostenlos abgegeben wird, seinerseits von den Beschränkungen des Urheberrechts befreit werden. Er könnte dann beispielsweise die Werke anderer zustimmungsfrei nutzen, auch wenn diese bislang durch das Urheberrecht geschützt wurden. Einerseits ergeben sich da verfassungsrechtliche Bedenken[2]. Andererseits ist aber auch klar, dass damit andere Urheber direkt geschädigt werden könnten: Ein Urheber könnte ein geschütztes Werk „dünn einpacken“[3] und fortan wäre das Gesamtwerk kostenlos erhältlich. Der Schaden für den Haupt-Urheber könnte beträchtlich sein, sofern dieser auf die Einnahmen angewiesen ist und sich bewusst gegen eine kostenfreie Abgabe entschieden hat.

Auch eine steuerliche Förderung wäre denkbar, z.B. über die Abziehbarkeit der Aufwendungen[4]. Problematisch daran ist, dass die Qualität des Werkes nicht beurteilt werden kann. Es kann aber nicht das Ziel sein, Arbeit zu belohnen. Leistung und der für die Gesellschaft entstehende Nutzen wären passendere Indikatoren. Zur Bewertung der Nützlichkeit dient aber in einer Marktwirtschaft gerade der Preis des Produktes. Ohne Preis kann man die Nützlichkeit nicht bewerten. Modelle wie bei der Parteienfinanzierung[5] wären bei Open-Source denkbar, würden aber ganz ohne Preis bzw. Benutzerregistrierung ebenfalls nicht funktionieren und wären missbrauchsanfällig. Auch damit scheidet also eine Förderung der kostenlosen Nutzungseinräumung aus praktischen Gründen aus.

1. Vgl. den schon zitierten „Browser-Krieg“ zwischen Microsoft und Netscape.
2. Das könnte eine Enteignung darstellen, vgl. Art. 14 GG.
3. Z.B. indem er noch einige bedeutungslose Zeilen hinzufügt.
4. Die Aufwendungen können typischerweise nicht steuerlich als Kosten geltend gemacht werden, da das Finanzamt aus der fehlenden Gewinnerzielungsabsicht Liebhaberei und damit steuerliche Nichtabziehbarkeit folgern wird. Eine Lösung könnte in der Gründung gemeinnütziger Vereine oder gGmbHs bestehen, wobei durch letztere auch ein eventuelles Haftungsproblem gelöst wäre.
5. Die Parteien erhalten pro Euro Spende bzw. pro für sie abgegebene Wahlstimme einen bestimmten Betrag vom Staat. Damit knüpft man an ihre Beliebtheit an.

3. Zusammenfassung

Weder ein Verbot noch die Förderung der kostenlosen Nutzungseinräumung sind im Urheberrecht möglich. Die momentane neutrale Haltung ist hingegen unproblematisch und lässt Spielraum in beide Richtungen.

III. Zusammenfassung

Open Source ist kein Widerspruch zu den wirtschaftlichen Grundannahmen des Urheberrechts. Diese sind auch nicht zu überdenken.

D. Taugliches Modell für Informationsproduktion

I. Generelles Modell

1. Open-Source als einziges und verbindliches Modell

Man könnte Open-Source zum einzigen und verbindlichen Modell der Softwareentwicklung erheben. Diesem Ziel entsprechend könnten die rechtlichen Rahmenbedingungen umgestaltet werden.

2. Schwachstellen des Open-Source-Modells

a) Deckt nur wenige Bereiche ab

Gemessen an der Gesamtzahl möglicher Anwendungsfelder und der Zahl bereits existierender Softwarelösungen ist Open-Source eher unbedeutend. Die Frage ist, ob sich das ändern wird. Es gibt viele Bereiche, in denen Open-Source sich auch in Zukunft kaum entwickeln wird, geschweige denn überlebensfähig ist. Insbesondere in Anwendungsgebieten mit hochkomplexen Aufgabenstellungen, die zudem noch aufwendig zu implementieren sind, aber nur einen kleinen Markt haben, wird Open-Source nicht bestehen, weil es das Durchschnittswissen und die Leistungsfähigkeit von Freizeitentwicklern überfordert. Gleichzeitig wäre durch den kleinen Markt nicht sichergestellt, dass mit Hardware und Dienstleistungen genügend Einnahmen zu erzielen sind, so dass sich auch kommerzielle Entwickler diesem Bereich nicht nähern würden.

Wenn man die gesamte Breite möglicher Anwendungsdomänen nimmt, ist das zentrale Problem, dass Anforderungskenner und Programmierer zueinander finden, besser noch in einer Person liegen müssen. Es nutzt nicht viel, wenn derjenige, der einen Fehler bemerkt, diesen überhaupt nicht selbst beheben kann, oder der mögliche Fehlerbeheber den Fehler gar nicht versteht. Letztlich ist das genau die übliche Problematik in herkömmlichen Softwareprojekten. Wesentliche Erfolge von Open-

Source-Software sind darauf zurückzuführen, dass Anforderungskenner und Entwickler die gleiche Person waren[1]. Über alle Anwendungsdomänen betrachtet ist diese Annahme unrealistisch.

b) Klammert unbeliebte Tätigkeiten aus

Es mag ein Zufall sein, dass in der Regel nur die ausführbaren Dateien und die Quelltexte veröffentlicht werden. Es drängt sich aber der Verdacht auf, dass die Arbeitsweise in vielen Open-Source-Projekten nicht dem Stand der Technik entspricht. Eine definierte Anforderungsanalyse, geordnetes Testmanagement oder das Schreiben der Entwicklungs- und Benutzerdokumentation sind regelmässig eher unbeliebte Tätigkeiten bei Softwareentwicklern. Oft werden diese sogar als unproduktiv abgetan[2]. Selbst wenn die viel beschworenen Open-Source-Freizeitentwickler über derartige Qualifikationen verfügten, wäre höchst fraglich, ob sie ihre Freizeit damit verbringen würden[3]. Auch die Darstellung des Basar-Modells von Raymond[4] scheint im Benutzer eher den nützlichen Tester zu sehen als den jenigen, der einen Nutzen aus der Anwendung ziehen soll. Daher ist wahrscheinlich, dass die Open-Source-Urheber aus Produktivitätsgründen den grössten Teil des Aufwandes in der Softwareerstellung beiseite schieben und sich nur der Codierung widmen. Auf Dauer kann das allerdings nicht funktionieren. Volkswirtschaftlich ist es auch alles andere als effizient, wenn der Benutzer als Tester betrachtet wird. Es läge deutlich näher, den Urhebern zwangsweise qualitätssichernde Massnahmen aufzuerlegen bzw. die Nutzer über Gewährleistung und Haftung vor Schaden zu bewahren, um die volkswirtschaftlichen Kosten niedrig zu halten, die beim Einsatz durch eine grosse Anzahl von Nutzern entstehen[5].

c) Kooperation und Verbreitung abhängig vom Internet

Ein entscheidendes Medium zur Kooperation der Hersteller von Open-Source ist das Internet. Zwar gab es auch früher bereits die Möglichkeit, Daten über alternative, privat organisierte Netze auszutauschen[6]. In die-

1. Das ist letztlich genau das Urbeispiel von Stallman: Dieser hatte Probleme mit der Druckersteuerung (Anforderung) und benötigte den Source, um das Programm selbst zu ändern (Entwickler), vgl. Williams S. 8. Weitere Beispiele sind Linux, Apache und Qt. Jeweils gehörten die Entwickler auch zu den Benutzern.
2. Zu Unrecht, schliesslich hängen Qualität und Nützlichkeit stark davon ab.
3. Bei professionellen Entwicklern macht der Gewährleistungs- und Haftungsausschluss Open-Source erst richtig interessant. Gesellschaftlich ist dies jedoch kontraproduktiv, da immer mehr Software zweifelhafter Qualität zum Nutzer kommt.
4. Vgl. Raymond S. 27; bei seiner Schilderung entsteht unweigerlich der Eindruck, die Teilnehmer des Basar-Modells seien „nützliche Idioten“.
5. Die Entwickler legen die Kosten dafür auf die Nutzer um. Jedoch bleiben langfristig nur die Entwickler im Markt, deren Produkte von Nutzern lizensiert wurden und die erfolgreich in der Vermeidung derartiger Ansprüche waren.

sen existierten selten Direktverbindungen, um die Kosten[1] für jeden Teilnehmer gering zu halten. Auf diese Weise entstehen deutlich höhere Verzögerungen in der Kommunikation. Zudem steigen die Kosten stark mit der zu übermittelnden Datenmenge an. Früher war Open-Source daher eher ein Phänomen im Umfeld von Universitäten, die Zugang zum Internet hatten. Die Open-Source-Autoren mussten die Kosten für die Infrastruktur nicht selbst tragen. Mittlerweile kann jeder einen günstigen Zugang zum Internet erhalten und auch eine eigene Website betreiben[2]. Mit zunehmender Regulierung des Internet könnte der Zugang erschwert oder zumindest der grenzüberschreitende Verkehr behindert werden[3]. In diesem Fall ist sowohl eine Hemmung der Entwicklung als auch der Distribution von Open-Source zu befürchten.

d) Softwareentwickler abhängig von wirtschaftlicher Entwicklung

Softwareentwickler müssen sich ernähren. Darüber hinaus sind sie möglichem Wohlstand sicher nicht mehr als andere Berufsgruppen abgeneigt[4]. Die Zeit, die ein Softwareentwickler für Open-Source-Projekte aufwenden kann, hängt direkt von seinen Einnahmequellen, seinem Finanzbedarf und der Risikobereitschaft ab. Hier sind zwei Gruppen zu unterscheiden: professionelle Entwickler und Hobbyentwickler.

aa) Professionelle Entwickler

Die professionellen Entwickler versuchen sich durch Softwareentwicklung zu ernähren, während die Hobbyentwickler andere Einnahmequellen nutzen. Ein angestellter Softwareentwickler verdient – in überwiegend herkömmlichen Unternehmen – im Schnitt 47.200 Euro pro Jahr[5]. Dies übersteigt durchaus den Durchschnittsverdienst[6]. In Zeiten

6. Beispielsweise das private Mailboxnetz Fidonet, vgl. http://fidonet.fidonet.org.

1. Einige Alternativ-Netze sind über normale Telefonverbindungen organisiert, bei denen die Teilnehmer sich in bestimmter Reihenfolge die Daten weitergeben. Spontane Direktverbindungen wären zwar möglich, würden sich aber auch in entsprechenden Telefongebühren für Ferngespräche z.B. in die USA niederschlagen.
2. So mag z.B. das Webhosting rund 10 Euro pro Monat incl. 2 GB Transfervolumen kosten, http://www.schlund.de/fwebhosting.php. Der Zugang kann über T-Online dsl flat für 25 Euro pro Monat erfolgen incl. privater Website, vgl. http://service.t-online.de/t-on/kund/anme/star/CP/cc-anmeldung.html.
3. Das könnte eine Konsequenz aus Sicherheitsbedenken, schwer verfolgbaren Straftaten und unterschiedlicher Rechtsprechung sein; vgl. http://www.heise.de/newsticker/data/fr-26.04.01-000 oder http://www.heise.de/newsticker/data/jk-31.03.02-003.
4. Auch wenn Raymond dies bestreitet. Durch den Börsengang von Red Hat stiegen seine Aktien immerhin auf $ 36 Mio., vgl. Williams S. 167.
5. Vgl. http://www.heise.de/newsticker/data/em-11.03.02-000; die Studie „Entgelt in der IT-Branche“ der IG Metall, 3. Erhebung 2001, S. 78-82, spricht von Durchschnittswerten zwischen 70.332 (SW-Ingenieur I) und 138.065 DM (Projektleiter), allerdings auf Basis einer 35-Stunden-Woche.
6. Dieser lag z.B. für männliche Angestellte im produzierenden Gewerbe im Jahr 2000 bei 3730 Euro, vgl. http://www.destatis.de/basis/d/logh/loghtab9.htm, andere Branchen liegen darunter, z.B. das Kreditgewerbe mit 2922 Euro.

schlechter wirtschaftlicher Aussichten ist anzunehmen, dass die Bereitschaft bei Nicht-Arbeitslosen zurückgeht, in der Freizeit Open-Source zu entwickeln. Dafür könnte sie bei arbeitslosen Softwareentwicklern steigen. Allerdings wird es typischerweise so sein, dass gerade nicht die besten Entwickler arbeitslos sind und dies ausreichend lange bleiben.

bb) Hobbyentwickler

Auch zu den Hobbyentwicklern ist anzumerken, dass sie Software nur erstellen werden, solange sie sich nicht zwingend um andere Dinge kümmern müssen. Wenn sie gut genug sind, werden sie das Hobby zum Beruf machen. Hinzu kommt, dass Hobbyentwickler eine geringere Bindung an die Projekte aufweisen, weil sie nicht wirtschaftlich von ihnen abhängig sind.

cc) Zusammenfassung

Insgesamt sind die wertvollsten Open-Source-Entwickler (mit dem höchsten Knowhow und der meisten Zeit) abhängig von der Unterstützung durch einen Distributor oder gleichwertigen Finanzier. Bleibt diese Unterstützung aus[1], wird die Software höchstens mittelmässig sein. Daran ändert auch die Behauptung nichts, dass gerade das beim Open-Source-Modell nicht passieren würde[2]. Denn es reicht nicht aus, wenn ständig willige Hände und Köpfe nachwachsen, diese müssen auch das richtige Qualifikationsprofil haben.

e) Distributoren abhängig von guten Börsenzeiten

Der bisherige Höhepunkt des Open-Source-Modells fiel etwa zusammen mit dem Höhenflug der sogenannten „New Economy“ und damit einhergehenden Höchstkursen für IT-Unternehmen. Es verwundert zumindest, dass Open-Source früher keinen so grossen Stellenwert und keine so bedeutende Massenwirkung erzielt hat[3]. Linux selbst ist über 10 Jahre alt[4]. In dieser Zeit hat es sich von 10.000 Zeilen Quelltext mit etwa 512 kByte auf rund 3,7 Millionen Zeilen mit über 130 MByte entwickelt. Und doch hat es erst in den letzten 3 Jahren ernsthafte Verbreitung über die Anwendung auf Servern hinaus gefunden. Das hat viel damit zu tun, dass Endbenutzer eine komfortable Oberfläche erwarten, wie sie etwa Mac OS X von Apple oder Windows XP von Microsoft

1. Z.B. weil die Liquidität des Finanziers oder sein Börsenkurs sinkt.
2. Weil beliebig viele Hände/Augen daran arbeiten, vgl. Raymond S. 30-32.
3. Seit Jahren sind Musikinstrumente und Videokameras günstig erhältlich. Damit sind viel einfacher Werke zu erstellen als mit Programmiersprachen. Trotzdem gibt es keine relevante Open-Source-Musik oder Open-Source-Filme. Das Verschenken und Verbreiten solcher Werke wäre schon lange möglich.
4. Vgl. http://www.heise.de/newsticker/data/odi-24.08.01-001.

bietet. Eine solche Oberfläche ist aber sehr aufwendig zu erstellen. Zwar gibt es die Unix-Oberfläche X11 schon seit 1986[1]. Diese war jedoch für Windows-gewohnte Benutzer kaum tauglich. Hinzu kamen Probleme bei der ursprünglich vergleichsweise aufwendigen X11-Programmierung[2]. Auch waren keine günstigen oder kostenlosen Office-Umgebungen erhältlich[3]. Für diese Produkte sind immense Vorinvestitionen erforderlich gewesen. Es drängt sich der Verdacht auf, dass diese nur zu Zeiten der Börseneuphorie finanzierbar waren. So hatte beispielsweise StarOffice in den vorherigen rund 15 Jahren seiner Existenz ein Nischendasein gefristet, bevor Sun Microsystems die Firma für über $ 70 Millionen übernahm, fortan für das Produkt kostenlos Lizenzen vergab und es zum Hoffnungsschimmer unter Linux avancierte[4]. Es scheint fraglich, ob Sun Microsystems heute noch einmal finanziell dazu in der Lage wäre[5].

Mit Verwunderung haben viele Linux-Befürworter zur Kenntnis genommen, dass Red Hat selbst einige Software-Patente auf Linux-Technologien angemeldet hat[6]. Es bleibt abzuwarten, ob diese in Zukunft gegen Mitbewerber eingesetzt werden, insbesondere wenn es der Firma Red Hat schlechter gehen sollte.

f) Abhängig von strategischen Interessen grosser IT-Konzerne

Die Distributoren sind eine wesentliche Quelle der Fortentwicklung von Open-Source-Projekten, da sie selbst Entwickler einstellen oder externe Entwickler bezahlen, damit sie ihrer Tätigkeit nachgehen können. Gleichzeitig sind die Distributoren jedoch abhängig von den Interessen grosser IT-Konzerne. So wird beispielsweise die schon 1992 gegründete SuSE Linux AG[7] von Intel, IBM und SAP finanziert[8]. Auch von AMD sind wesentliche Aufträge gekommen[9]. Oracle zählt zu den Technologie-Partnern[10]. Von SuSE sind Liquiditätsprobleme bekannt,

1. Vgl. http://www.x.org.
2. Mittlerweile gibt es genügend Bibliotheken, die eine Abstraktion bieten, z.B. GTK, http://www.gtk.org, oder Qt, http://www.trolltech.com.
3. Heute mag man StarOffice, http://wwws.sun.com/software/star/staroffice/5.2/, OpenOffice, http://www.openoffice.org, oder ApplixWare, http://www.vista-source.com/products/axware/, nehmen.
4. Vgl. http://www.heise.de/newsticker/data/db-18.01.01-000; StarDivision selbst hat seit 1998 für den Privatgebrauch kostenlose Lizenzen vergeben; vgl. http://www.heise.de/newsticker/data/db-13.11.98-003.
5. Vgl. http://www.heise.de/newsticker/data/jk-02.05.02-005.
6. Vgl. http://www.heise.de/newsticker/data/daa-25.05.02-002.
7. Nach Entlassungen hat sie noch 380 von 550 Mitarbeitern.
8. Vgl. http://www.heise.de/newsticker/data/odi-28.09.01-000; offizielle Informationen liegen nicht vor, da die SuSE AG auf ihren Webseiten ihre Anteilseigner nicht nennt. Ein weiterer Finanzier ist die Deutsche Bank.
9. Linux-Portierung auf 64-Bit-Prozessor Sledgehammer; vgl. http://www.heise.de/newsticker/data/ju-15.08.00-001.

zudem hofft die Firma im ersten Halbjahr 2002 endlich in die Gewinnzone zu kommen, im Jahr 2001 lag der Umsatz bei 40 Millionen Euro[1]. All das zeugt nicht gerade vom wirtschaftlichen Erfolg eines Unternehmens, das auf eigenen Beinen steht[2]. SuSE dürfte nach Red Hat weltweit immerhin der zweitgrösste Linux-Distributor sein. In Deutschland ist kein grösseres Unternehmen bekannt, das sein Schicksal derart auf das Open-Source-Modell baut[3]. Die eigentliche Distribution von Linux an Privat- und Kleinkunden scheint jedoch wenig Geld einzubringen, so dass bei SuSE ein Schwenk hin zur Unternehmens-IT erfolgte[4].

g) Qualität ist nicht im Interesse der Open-Source-Anbieter

Die Open-Source-Hersteller und Distributoren setzen darauf, dass sie durch Hardware-Verkauf, kostenpflichtige Zusatzsoftware oder Dienstleistungen ihre Leistung refinanzieren können. Daraus ergibt sich aber, dass sie kein Interesse an Software haben können, die auf Standard-Hardware läuft, die schon jeder Nutzer besitzt. Wenn die kostenlose Open-Source-Version alles Benötigte enthält, lizensiert niemand die Zusatzprogramme. Einfach zu bedienende Software würde Schulungen überflüssig machen. Wenn die Software stabil ist, benötigt man keinen Support-Vertrag[5]. Ist sie funktional vollständig, werden weniger Nutzer Erweiterungen in Auftrag geben. Bei einfach zu verstehenden und gut dokumentierten Quelltexten wären die Entwickler nahezu beliebig austauschbar und Dritte könnten sich problemlos einarbeiten. All das läuft den Finanzierungsmechanismen der Open-Source-Produzenten diametral entgegen. Schliesslich möchte man sich über Schulung, Support, Beratung, Fehlerbeseitigung, Erweiterung und Hardware finanzieren. Das Geschäftsmodell ist also alles andere als unkritisch. Wenn schon herkömmliche Softwarefirmen, die immense Lizenzeinnahmen haben, mangelnde Qualität zur Generierung von Umsätzen nutzen[6], so muss

10. Vgl. http://www.heise.de/newsticker/data/anw-23.12.01-000.
1. Vgl. http://www.heise.de/newsticker/data/anw-23.12.01-000.
2. Die Zahlen von SuSE sind kein Einzelfall: Caldera machte im 1. Quartal 2001 noch $ 11 Mio Verlust bei einem Umsatz von $ 17,9 Mio, vgl. http://www.heise.de/newsticker/data/odi-28.03.02-001. Red Hat schreibt nur schwarze Zahlen, wenn man Sonderbelastungen herausrechnet, sonst waren es allein im dritten Quartal 2001 $ 15 Mio. Verlust, vgl. http://www.heise.de/newsticker/data/odi-20.12.01-000.
3. Der Distributor Caldera hat nur 40 Mitarbeiter, plant zudem Entlassungen, vgl. http://www.heise.de/newsticker/data/odi-28.03.02-001.
4. Vgl. http://www.heise.de/newsticker/data/anw-23.12.01-000.
5. Linus Torvalds veröffentlichte früher pro Tag mehrere Kernelversionen. Das dürfte ganz das Gegenteil von Stabilität und Effizienz sein und erzeugt eher den Eindruck von „try and error"-Programmierung. In herkömmlichen Entwicklungsprozessen wird das dadurch verhindert, dass vor jedem Release ein Regressionstest durchzuführen ist, um zu verhindern, dass die Änderungen die bisherige Funktionalität beeinträchtigen und hinter den bisherigen Stand zurück werfen.

dies in Zukunft erst recht bei Open-Source-Firmen befürchtet werden.

h) Strukturelle Ungleichheit von Einzelentwicklern und Konzernen

Grundsätzlich könnten die Urheber von Open-Source ihre Investitionen durch Hardware und Dienstleistungen refinanzieren. Diese Annahme muss man allerdings relativieren, wenn man einen Einzelentwickler mit einem Konzern wie z.B. EDS vergleicht. EDS hat etwa 40.000 Mitarbeiter[1], die zum Teil hoch spezialisiert sind, während ein einzelner Entwickler derartige Spezialisierungen kaum erreichen kann[2]. Sofern beide auf dem Markt um Serviceverträge werben, dürften die Grosskonzerne deutlich bessere Karten haben, allein schon aufgrund ihrer professionellen Aquise und Vertragsabwicklung[3]. Aber auch die höhere Redundanz verstärkt diesen Effekt[4]. Eine Möglichkeit für die Entwickler besteht darin, sich von grossen Firmen einkaufen zu lassen. Letztlich unterscheidet sich dann das Open-Source-Modell für diese Entwickler nicht mehr von herkömmlichen Lizenzen. Die oft beschworene „Community“ hilft hier nicht um sicherzustellen, dass die Einnahmen fair auf die Urheber und Dienstleister verteilt werden. Open-Source ist daher eher als Umsatzbeschaffer für grosse IT-Dienstleister und Hardwareverkäufer[5] zu bewerten.

i) Entwickler enttäuscht, weil andere verdienen

Sofern ein Entwickler nur geringe Investitionen getätigt hat, wird seine Frustration nicht sehr gross sein, wenn andere die Früchte ernten. Wenn aber mehrmonatige Arbeit geleistet wurde, ist das schwer einsehbar. In der Folge könnte der Entwickler gar keine Software mehr unter eine Open-Source-Lizenz stellen. Die Entwickler müssten also angemessen am Erfolg ihrer Werke beteiligt werden[6]. Leider lässt sich oft gar nicht feststellen, wer überhaupt in welchem Umfang zum Erfolg beigetragen hat, da bei der Basar-Methode möglicherweise hunderte oder tausende

6. Man sagt Microsoft nach, diese Technik einzusetzen, indem man wenig Wert auf Qualität legt und damit Benutzer zum Lizensieren der Nachfolgeversion bringt, in der Hoffnung, dass dort die betreffenden Funktionen stabil sind.
1. Vgl. http://www.eds.de.
2. Einzelentwickler oder kleine Gruppen können derartige Spezialisierungen nicht leisten, weil das Risiko gross ist, dass gerade dieses Gebiet innerhalb des Zyklus von 18 Monaten bis zur Veralterung keine ausreichenden Einnahmen sichert.
3. Man muss berücksichtigen, dass diese Konzerne oft schon „den Fuss in der Tür haben“, während der Einzelentwickler die potentiellen Kunden nicht einmal kennt, und aufgrund von Einkaufsrichtlinien für Anbieter auch keine Chance hat, überhaupt als Lieferant anerkannt zu werden.
4. Wenn der Entwickler krank wird, kann er kein Geld mit Service verdienen. Er kann auch nur eine Schulung gleichzeitig geben bzw. mit einem Kunden zur Erfüllung des Supportvertrages gleichzeitig telefonieren.
5. Vgl. beispielsweise IBM, Red Hat, SuSE.
6. Das war eine Kernüberlegung der Änderung des Urhebervertragsrechts.

Entwickler beteiligt sind und es keine formellen Aufzeichnungen gibt. Prinzipiell leben die Distributoren von der Verbreitung der Arbeitsergebnisse fremder Entwickler. Sie finanzieren zwar einige wichtige Entwickler, um ihnen die Arbeit an Open-Source-Projekten zu ermöglichen. Jedoch müssen die Distributoren dabei im Rahmen ihrer eigenen finanziellen Möglichkeiten bleiben. Das Open-Source-Modell versucht die Benutzer zu Mitentwicklern umzufunktionieren[1]. Auch die Benutzer können enttäuscht sein, wenn sie bemerken, dass andere in grossem Maße von ihrer Arbeit profitieren[2].

j) Dokumentation kaum verfügbar

So schön die Verfügbarkeit der Quelltexte auch ist, in den wenigsten Fällen ist diese hinreichend. So ist seit Jahren gängige Rechtsprechung, dass zumindest bei Individualsoftware eine Entwicklungs-, Wartungs- sowie eine Benutzerdokumentation erforderlich ist. Sieht man sich die Linux- oder Mozilla-Quellen[3] an, so sind diese praktisch undokumentiert. Selbst für so komplexe Dinge wie den Linux-Kernel ist keine aktuelle Dokumentation erhältlich[4]. Die Verwendung von UML-Diagrammen[5] scheint in Open-Source-Projekten generell nicht beliebt zu sein. Dokumentation zur Qualitätssicherung, z.B. Testfallbeschreibungen und Reviewprotokolle, ist in der Regel überhaupt nicht vorhanden. Das ist zwar einerseits verständlich[6], andererseits aber höchst gefährlich. Schliesslich können in Open-Source-Projekten häufig die beteiligten Entwickler schnell wechseln. Sie unterliegen keinerlei Zwängen, die in Firmen immer noch vorhanden sind[7].

Zwei Erklärungen sind denkbar: Entweder erstellen bzw. veröffentlichen die Entwickler diese Dokumentation absichtlich nicht oder verwenden sie aus Unkenntnis nicht. Beides hinterlässt ein ungutes Gefühl. Ohne derartige Dokumentation ist die Entwicklung in hohem Maße per-

1. Vgl. Raymond S. 27.
2. So haben freiwillige Forumsbetreuer gegen AOL geklagt (USA). Sie argumentierten, AOL habe ihre, wenn auch freiwilligen, Leistungen als Produkt verkauft und begehrten Zahlung eines Mindestlohns, als AOL einen Jahresumsatz von $ 7 Milliarden auswies, von dem etwa ein Drittel auf ihren kostenlosen Leistungen basierte, vgl. http://www.heise.de/newsticker/data/anm-08.02.01-001.
3. Erhältlich unter http://www.kernel.org bzw. http://www.mozilla.org.
4. „The Linux Kernel Module Programming Guide“ stammt aus dem Mai 1999, „Linux Kernel 2.4 Internals“ aus dem August 2001, vgl. http://www.tldp.org/guides.html.
5. Vgl. das Beispiel zum Sequenzdiagramm in http://www.omg.org/cgi-bin/doc?formal/01-09-67, Abschnitt 3.60.4 oder Zustandsdiagramm in 3.74.2.
6. Weil den meisten Entwicklern Dokumentation und Qualitätssicherung keine Freude macht, in ihrer Freizeit schon gar nicht.
7. Natürlich können auch angestellte Entwickler kündigen, die Hemmschwelle ist aber höher als bei einem Open-Source-Projekt.

sonenabhängig. Fällt ein Entwickler aus, geht sein undokumentiertes Wissen über das Projekt verloren.

Es ergibt sich der Eindruck, dass die Verfügbarkeit der Quelltexte stark überbewertet wird. Diese sind nur ein kleiner Teil des Gesamtproduktes. Mindestens ebenso wichtig wäre die Verwendung eines anerkannten Erstellungsprozesses und formale Dokumentation der Arbeitsergebnisse, damit sachkundige Dritte tatsächlich in angemessener Zeit den Quelltext verstehen und ändern können. Gerade das liegt aber kaum im Interesse der Entwickler. Sie hätten (unbeliebte) Arbeit damit und würden sich zusätzlich noch austauschbar machen. Gerade in einer Zeit, in der Software sehr schnell veraltet, erhält man daher den Eindruck, dass die Open-Source-Entwickler mit dem Quelltext nur etwas veröffentlichen, was Dritten kaum nützt. Wer 1,3 Millionen Zeilen[1] Quelltext vor sich hat, wird ohne Dokumentation eine lange Zeit brauchen, um Fehler zu finden oder das Programm zu erweitern, selbst wenn er die benutzte Programmierumgebung kennen sollte und ihm die verwendete Hardware zur Verfügung steht.

k) Zusammenfassung

Das Open-Source-Modell hat viele bedeutende Schwachstellen. Open-Source ist abhängig von der allgemeinen wirtschaftlichen Entwicklung und von grossen Geldgebern. Auf lange Sicht entstehen Zweifel an der Qualität der Arbeitsmethoden und -ergebnisse. Es ist auch nicht erkennbar, wie diese Schwachstellen beseitigt werden könnten, ohne das Modell insgesamt zu beschädigen.

3. Rechtliche Probleme

Eine der am weitesten verbreiteten Konstellationen dürfte der Download von Open-Source-Software aus dem Internet sein. Es ist strittig, welcher schuldrechtliche Vertragstyp dafür in Frage kommt und ob ein schuldrechtlicher Vertrag überhaupt benötigt wird[2].

a) Anwendbares Recht

Unklar ist, welches Recht überhaupt zur Anwendung kommen soll. Es kommt eine konkludente Vereinbarung US-amerikanischen Rechts in Frage. Sofern man diese ablehnt[3], kommt deutsches Recht in Frage oder

1. Vgl. http://www.heise.de/newsticker/data/odi-24.08.01-001.
2. Vgl. die Diskussion der GPL bei Grützmacher S. 84 ff. Grundsätzlich ist eine Unterscheidung der Distributionswege (Download aus dem Internet direkt vom Urheber, CD-Kauf vom Distributor usw.) wie bei Jaeger/Metzger notwendig; darauf wird hier verzichtet, da es nur um einen Problemaufriss geht.
3. Vgl. ablehnend Sester S. 802 und Grützmacher S. 86 II.1.

das jeweils nationale Recht der Urheber. Problematisch ist auch die in der Regel nur englischsprachige Fassung der Lizenzen[1]. Man mag argumentieren, dass der Nutzer von Software eben Englisch verstehen müsse. Für juristische Formulierungen wie die GPL ist das gegenüber Verbrauchern jedoch sicher zu viel verlangt[2].

b) Auftrag

Man könnte einen Auftrag i.S.d. § 662 BGB annehmen[3]. Zu recht wird jedoch kritisiert[4], dass der Urheber dann einen Auftrag des Nutzers ausführen müsste und dies nicht passt, da der Urheber nicht fremde, sondern eigene Interessen verfolgt[5]. Zwar ist ein Eigeninteresse nicht schädlich[6]. Allerdings darf bezweifelt werden, dass es sich bei der Erstellung von Open-Source-Software typischerweise um ein vom „Auftraggeber übertragenes Geschäft" im Sinne einer Geschäftsbesorgungsmacht *in fremdem Rechtskreis* handelt[7]. Hinzu kommt, dass die Annahme eines Auftrages (§ 662 BGB) die Abgabe eines Angebotes voraussetzt. Das Zurverfügungstellen der Open-Source-Software im Internet kann man als invitatio ad offerendum werten[8]. Der Nutzer müsste also das Angebot abgeben, auch damit die Annahme durch den Urheber geschehen könnte. Dieses müsste den Umfang des Auftrages ausreichend beschreiben. Bei genauer Betrachtung liegt jedoch diesbezüglich überhaupt keine Kommunikation vom Nutzer zum Urheber vor, die man als Übertragung eines Auftrages werten könnte[9]. Schlimmer noch: Wenn man von einer folgenden Annahme durch den Urheber ausgeht (z.B. konkludent durch Senden der Datei), so verkehrt man gerade den Sinn der Betrachtung als invitatio ad offerendum, denn der Urheber würde nun automatisiert in eine Vielzahl von Geschäftsbesorgungsverträgen einwilligen, obwohl er den mutmasslichen Auftraggeber nicht kennt und in vielen Fällen nicht einmal ermitteln kann[10]. Nimmt man den-

1. Vgl. Grützmacher S. 88 III.6.
2. Grützmacher S. 88 III.6 verlangt eine deutsche Fassung.
3. Vgl. Marly Rn 318, bezgl. der sehr ähnlichen Freeware. Kritisch: Sester S. 800.
4. Vgl. Jaeger/Metzger, 4. Kapitel A I 3.
5. Bei genauer Betrachtung mag man schon daran zweifeln, dass er überhaupt „tätig *wird*". Typischerweise ist die Software längst fertig und wird nicht etwa erst auf Anforderung des Nutzers hin erstellt.
6. Vgl. Erman/Ehmann, vor § 662 Rn 51, danach können „Eigen- und Fremdinteresse stufenlos von Null bis Eins gemischt sein", erfordert aber „ein Minimum an Fremdinteressenwahrung".
7. Vgl. Erman/Ehmann, vor § 662 Rn 51.
8. Vgl. Erman/Hefermehl § 145 Rn 10 zum Warenangebot in Datennetzen.
9. Sofern es überhaupt zu einer Kommunikation mit dem Urheber kommt, findet diese rein auf der TCP-Ebene statt und wird kaum als hinreichend bestimmter Vertragsantrag hinsichtlich einer Beauftragung des Urhebers zu werten sein. Typischerweise umfasst diese „Erklärung" auf Ebene der üblichen HTTP- oder FTP-Protokolle lediglich die Aufforderung „Schick mir die Datei xyz".

noch einen Auftrag an, verbleiben Risiken.

aa) Haftung

Der Auftragnehmer haftet für die ordentliche Erledigung des Auftrages. Mangels diesbezüglicher Kommunikation kann man auch nicht von einer Abbedingung ausgehen. Vom Verständnis des Open-Source-Gedankens her wird man jedoch feststellen, dass der Urheber gerade keine Haftung übernehmen möchte.

bb) Prüfungs-, Warn- und Rückfragepflichten

Der Urheber hätte in diesem Modell auch eine Prüfungs-, Warn- und Rückfragepflicht[1]. Diese könnte er aber mangels Kenntnis des Auftraggebers in typischen Fällen gar nicht erfüllen[2]. Auch dies passt damit nicht zur Open-Source-Realität.

cc) Anspruch auf Aufwendungsersatz

Ferner hätte der Urheber einen Anspruch auf Aufwendungsersatz aus der Geschäftsbesorgung. Damit dürfte wiederum der mutmassliche Auftraggeber kaum einverstanden sein, denn der möchte ja gerade kostenlos an die Software gelangen. Es bedürfte also einer weiteren Konstruktion, um die Unentgeltlichkeit sicher zu stellen.

dd) Recht zur jederzeitigen Kündigung

Nun hätte der Auftragnehmer gemäss § 671 I BGB überdies ein Recht zur jederzeitigen Kündigung des Auftrages. Schwierigkeiten bereitet die Definition, worin der Auftrag besteht und wann er erledigt ist. Problematisch könnte sein, dass der Auftragnehmer nicht zur Unzeit kündigen darf. Die Open-Source-Anbieter möchten aber keinerlei Verpflichtungen haben, und schon gar nicht, wenn es „ernst wird".

ee) Auskunfts-/Rechenschaftspflicht

Darüber hinaus hätte der Auftraggeber gegenüber dem Urheber auch eine Auskunfts- und Rechenschaftspflicht. Diese wird er aber regelmässig nicht eingehen wollen, da er grundsätzlich jegliche Ansprüche des Nutzers ablehnt, die über die reinen urheberrechtlichen Nutzungsrechte hinaus gehen.

ff) Zusammenfassung

Die Annahme eines Auftrages wäre realitätsfern.

---

10. Die Rückverfolgung des TCP-Protokolls hin zu einer natürlichen Person ist äusserst aufwendig, wenn nicht sogar unmöglich (z.B. im Falle mehrerer Benutzer eines Internet-Zugangs, öffentliche Terminals), sofern es sich um die Menge der Programmnutzer handelt. Bei den Entwicklern sieht das etwas anders aus.

1. Vgl. Erman/Ehmann § 665 Rn 14 ff.
2. Man könnte höchstens annehmen, der Auftrag sei sowieso nicht mehr kündbar, da er bereits mit dem Download erledigt sei.

c) Schenkung

Man könnte das schuldrechtliche Verhältnis von Urheber und Nutzer als Verhältnis von Schenker zu Beschenktem beurteilen[1]. Aus dem Schenkungsrecht ergeben sich jedoch einige Probleme.

aa) Haftung

Höchst problematisch ist der Ausschluss der Haftung durch die GPL. Die GPL ist grundsätzlich als AGB zu betrachten[2]. In AGB kann die Haftung für grobe Fahrlässigkeit nicht abbedungen werden[3]. Genau das versucht aber die GPL[4]. Erst recht ist kritisch, dass (vermutlich unbewusst) sogar die Haftung für Vorsatz abbedungen werden soll[5]. Letzteres ist in keinem Fall akzeptabel.

bb) Rückforderung möglich

Nimmt man eine Schenkung an, so wäre der Schenker unter bestimmten Bedingungen berechtigt, das Geschenk zurückzufordern. Der Beschenkte könnte dies dadurch abwenden, dass er für den Unterhalt des Schenkers sorgt. Es bliebe zu untersuchen, inwiefern die praktische Vereitelung der Rückforderungsmöglichkeiten eine Gesetzesumgehung darstellt. Die Rückforderung hat schliesslich auch den Zweck, der Allgemeinheit den Rückgriff auf die Beschenkten zu ermöglichen, um keine Sozialleistungen für den notleidenden Schenker tragen zu müssen[6].

cc) Widerruf wegen grobem Undank möglich

Der Schenker könnte die Schenkung wegen grobem Undank widerrufen. Praktisch relevant könnte dies z.B. sein, wenn ein Nutzer der Software später öffentlich die Qualität der Open-Source-Software des Urhebers kritisiert.

dd) Anspruch auf Beseitigung von Rechtsmängeln möglich

Der Beschenkte hätte Anspruch auf die Beseitigung von Rechtsmängeln, die dem Geschenk anhaften. Dies hätte fatale Konsequenzen im Zusammenhang mit Softwarepatenten und könnte dazu führen, dass der Urheber auch noch Patentlizenzen erwerben und den Beschenkten zur

1. Vgl. Jaeger/Metzger, Vorbemerkung 4. Kapitel.
2. vgl. dazu ausführlich Sester S. 804, teilweise dagegen Koch S. 333.
3. Vgl. Grützmacher S. 88 III.6 und Zahrnt/Erben S. 5.
4. Vgl. Jaeger/Metzger S. 181, GPL „NO WARRANTY“, 11 und 12, auch wenn sie sich letztlich nationalem Recht beugt. Sester argumentiert sehr grosszügig (S. 806) und sieht die Grenze bei der groben Fahrlässigkeit und „evidenten Verstössen gegen fundamentale Regeln der Programmierkunst“ (S. 807).
5. Zwar sind die Klauseln eingeschränkt durch „to the extend permitted by applicable law“, jedoch gibt es Zweifel bezüglich der Wirksamkeit einer solch weiten Formulierung in AGB.
6. Zwar wäre auch bei einer Handschenkung an einen Unbekannten die Rückforderung vereitelt. Jedoch liegt der Fall bei Open-Source-Software anders, da die Nutzer bzw. Distributoren der Software oft durchaus bekannt sind.

Verfügung stellen müsste. Eine vertragliche Abbedingung per Open-Source-Lizenz könnte unwirksam sein.

ee) Handschenkung oder Formbedürftigkeit

Für eine Schenkung kommen nur zwei Formen in Betracht: Die Handschenkung oder Vertragsschenkung. Letztere kann unmöglich vorliegen, da formbedürftig. Die Handschenkung ist allerdings fragwürdig, da sie erkennbar auf die Übertragung körperlicher Objekte gerichtet ist.

ff) Entreicherung des Schenkers

Die Schenkung setzt grundsätzlich eine Entreicherung des Schenkers voraus. Auch dies verdeutlicht, dass das gesetzliche Leitbild auf körperliche Objekte zugeschnitten ist, bei denen durch die Weitergabe an den Beschenkten automatisch eine Entreicherung des Schenkers eintritt. Bei Software ist diese Entreicherung problematisch, denn der Urheber kann die Software weiter nutzen (er stellt dem Beschenkten nur eine Kopie des entsprechenden Bitmusters zur Verfügung). Auch wenn man die Lizenz anstelle der Daten betrachtet, so wird regelmässig eine nicht ausschliessliche Lizenz gewährt. Damit bleibt der Urheber selbst auch berechtigt, das Werk zu nutzen. Auch hier kann man also keine Entreicherung entdecken.

gg) Zusammenfassung

Eine Schenkung deckt sich nicht mit dem Willen der Parteien, passt nicht zur gesetzlichen Intention und führt zu absurden Ergebnissen.

d) Schlichte Gestattung

Laut Jaeger/Metzger setzt jede urheberrechtliche Nutzungsbefugnis eine schuldrechtliche Vereinbarung voraus[1]. Eine schuldrechtliche Vereinbarung ist zwar sehr oft anzutreffen, aber keineswegs notwendig. Es kommt auch eine schlichte einseitige Einwilligung in die Nutzung in Frage[2], sofern man zwischen der lizenzrechtlichen Beziehung und einem (ggf. gar nicht existenten) Softwareüberlassungsvertrag trennt[3]. Dafür spricht auch § 69 c UrhG, der besagt, der Rechtsinhaber könne die aufgezählten Handlungen „gestatten“, oder § 69 d I UrhG mit der Formulierung „Zustimmung des Rechteinhabers“. Eine Zustimmung kann auch ohne schuldrechtlichen Vertrag erteilt werden[4]. Durch einseitige, an die Öffentlichkeit gerichtete Erklärung des Urhebers, kann

1. Vgl. Jaeger/Metzger, Vorbemerkung 4. Kapitel.
2. Vgl. Schricker/Schricker Vor §§ 28 ff. Rn 27.
3. Vgl. Grützmacher S. 89 III.12.
4. Als rechtsgeschäftsähnliche Handlung, Schricker/Schricker vor §§ 28 ff. Rn 27.

dieser die Nutzung der Software gestatten[1]. Damit entfällt die Rechtswidrigkeit[2]. Dies dürfte den Interessen beider Seiten am ehesten entsprechen. Der Urheber hat keinerlei weitergehende Verpflichtungen gegenüber dem Nutzer. Der Nutzer darf die Software einsetzen. Ihm ist klar, dass er keine weitergehenden Ansprüche gegen den Urheber hat, gerade weil typischerweise kein Softwareüberlassungsvertrag mit diesem existiert.

e) Verweis auf mangelnde Aktivlegitimation

Ein grosses praktisches Problem könnte die Durchsetzung der Urheberrechte von Seiten der Open-Source-Entwickler darstellen, da ein Gemenge von Ansprüchen unterschiedlicher Anspruchsteller entsteht, welches zudem ggf. noch verschiedenen nationalen Urheberrechtsausprägungen unterliegen kann[3]. So müssten z.B. bei Geltendmachung von Unterlassungsansprüchen am heutigen Linux-Kernel sämtliche Urheber bezeichnet werden, was so gut wie unmöglich sein dürfte[4]. Darin könnte der Grund liegen, dass bislang kein einziger Prozess bekannt geworden ist[5]. Hier wäre es sinnvoll, nach rechtlichen Lösungen zu suchen, die eine Rechtsdurchsetzung seitens der Urheber erleichtern. Andererseits würde sich somit die Frage nach dem Wert von Open-Source-Lizenzen stellen. Man kann andererseits davon ausgehen, dass der Grossteil der Beiträge nur von sehr wenigen Personen stammt, besonders die Änderungen am Quelltext[6], so dass es bei gravierenden Verletzungen möglich erscheint, mit vertretbarem Aufwand alle Urheber an einen Tisch zu bekommen[7]. Dafür spricht auch die Untersuchung von Ghosh/Prakash, wonach nur 8,3 % der Beiträge nicht eindeutig einem Urheber zugeordnet werden konnten[8]. Da regelmässig Versionskon-

1. Vgl. Schricker/Loewenheim § 69 c Rn 3 für Public-Domain-Software.
2. Vgl. Schricker/Schricker vor §§ 28 ff. Rn 27 und Schricker/Wild § 97 Rn 19; Möhring/Nicolini § 97 Rn 68.
3. Vgl. Koch, S. 279 mit einer sehr ausführlichen Darstellung.
4. Vgl. Koch, S. 279, er spricht von rund 200 Personen.
5. Vgl. Koch, S. 279.
6. Vgl. Lerner/Tirole S. 11 f., danach stammen von den häufigsten 10 % der Urheber insgesamt 72 % des Codes, von den häufigsten 20 % schon 80 % des Codes. Die Statistik dürfte noch deutlicher ausfallen, wenn man berücksichtigt, dass von 6 Beiträgen nur einer aus Code besteht und die restlichen 5 nur Fehlerberichte sind.
7. Die Free Software Foundation weist auch darauf hin, dass nur die Urheber selbst gegen Verletzungen vorgehen können, wobei man sie aber gerne unterstützen werde, vgl. http://www.fsf.org/licenses/gpl-violation.html.
8. Vgl. http://firstmonday.org/issues/issue5_7/ghosh/index.html; auch über die Statistik, zu welchen Projekten der Autor beigetragen hat, dürfte er sich schnell identifizieren lassen, z.B. http://orbiten.org/ofss/codd-render.cgi?action=author&key=5195, da ihn vermutlich ein Mitautor per Mailadresse kennt. Als Fallbeispiel liess sich Gordon Matzigkeit allein durch Eingabe des Namens bei http://www.google.de sehr schnell der Mailadresse gord@m-tech.ab.ca zuordnen. Es ist anzunehmen, dass die dahinter stehende Firma M-Tech über seine Anschrift verfügt. Auch für weitere Namen liess sich dieses Vorgehen wiederholen.

trollsysteme verwendet werden, dürfte der Anteil der identifizierbaren Urheber in Zukunft eher noch steigen[1]. Ferner wäre zu prüfen, wie substantiiert die mangelnde Aktivlegitimation darzulegen ist. Sobald der Beklagte sich darauf beruft, müsste er vortragen, dass es noch weitere Urheber für das Werk gibt, und damit steigt die Wahrscheinlichkeit, eben diesen Urheber ausfindig zu machen und seine Prozessvollmacht zu erhalten.

f) Wegfall der Lizenz bei Unwirksamkeit einzelner Passagen

Teilweise wird in der Literatur die Frage gestellt, inwiefern die GPL in Deutschland den Nutzer überhaupt zur Verbreitung der Software berechtigt. Die Einschränkungen bezüglich der Gewährleistung und Haftung sowie eventuell benötigten Patentlizenzen könnten aus der Lizenz selbst und durch ihre Einordnung als AGB[2], dazu führen, dass aus rechtlichen Gründen gar keine Verbreitung gestattet ist[3].

g) Zusammenfassung

Die rechtliche Einordnung von Open-Source-Software ist umstritten. Es ist auch nicht naheliegend, wie eine sachgerechte Abwägung zwischen den Interessen der Hersteller[4] und Verbraucher[5] vorgenommen werden kann. Plausibel ist die Annahme einer schlichten urheberrechtlichen Gestattung, während überhaupt kein Softwareüberlassungsvertrag, weder in Form von Schenkung noch Auftrag, vorliegt. Zumindest bei grossen Open-Source-Projekten mit vielen Urhebern spricht einiges dafür, dass die Urheber ihre Rechte aus praktischen Erwägungen nicht oder nur unter grossen Mühen[6] durchsetzen können. Auch gibt es gute Argumente dafür, dass derzeitige Open-Source-Lizenzen in Deutschland aufgrund der Rechtslage gar keine Weiterverbreitung erlauben, was entsprechende Konsequenzen für die Distributoren hätte. Auch eine Erschöpfung des Verbreitungsrechtes könnte die GPL-Konstruktion zum Einsturz bringen[7]. Es könnte auch im Interesse des Gesetzge-

1. So könnte z.B. allein die Protokollierung der zum Einstellen des Quelltextes verwendeten IP-Adresse schon zum Urheber führen, da nicht anzunehmen ist, dass sie absichtlich die IP-Adresse verfälschen oder schwer rückverfolgbar machen. Zu einer IP-Adresse kann man z.B. mit http://www.ripe.net/perl/whois, http://ws.arin.net/cgi-bin/whois.pl oder http://www.apnic.net/apnic-bin/whois.pl zuammen mit der Mailadresse mit einer hohen Erfolgsquote die Person ermitteln.
2. vgl. dazu ausführlich Sester S. 804.
3. Vgl. GPL 7.: „If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the program at all."
4. Z.B. auf vollständigen Haftungs- und Gewährleistungsausschluss.
5. Z.B. auf Haftung für grob fahrlässige und erst recht vorsätzliche Schädigung.
6. Z.B. das gesamte Prozesskostenrisiko.
7. vgl. Koch S. 336, der Nutzer dürfte gegen Vergütung und ohne Source verbreiten.

bers liegen, der GPL zur vollen Geltung zu verhelfen, da die Exekutive mittlerweile selbst in grossem Umfang Open-Source einsetzt.

4. Zusammenfassung

Aufgrund der vielen Nachteile und Probleme ist Open-Source als einziges und verbindliches Modell nicht geeignet.

II. Effiziente, dezentralisierte Informationsproduktion

Open-Source könnte effizienter und dezentralisierter sein als die herkömmliche Softwareentwicklung.

1. Effizienter

Zur Untersuchung der Annahme, Open-Source sei effizienter, so kann man verschiedene Aspekte unterscheiden.

a) Produktion

aa) Wiederverwendung

Die Erstellung von Software könnte im Open-Source-Modell effizienter sein, wenn es einen höheren Wiederverwendungsgrad gäbe. Bei Closed-Source oder Shared-Source kann nicht jeder, der möchte, einfach Teile des Sourcecodes übernehmen, sie ggf. verbessern oder als Ausgangsbasis für eigene Software nehmen.

bb) Konzentration auf kleinen Kreis

Die Konzentration des Sources auf einen kleinen Kreis in herkömmlichen Modellen hat auch Vorteile. Es ist deutlich effizienter, wenn sich wenige mit dem Source auskennen (z.B. die Entwickler des Herstellers) und als Ansprechpartner für Nutzer bereitstehen. Dass sich hunderte von Menschen mit dem Source ggf. nur für eine kurze Zeit beschäftigen überzeugt unter dem Gesichtspunkt Effizienz nicht. Eine höhere Spezialisierung ist grundsätzlich effizienter, und dazu bedarf es einer ausreichenden Konzentration. Auch für die Nutzer ist dies von Vorteil, da sie sicher sein können, dass der Hersteller ggf. der einzige ist, der Support leisten kann und sich das deshalb auch für ihn lohnt. Bei Open-Source könnten wirtschaftlich stärkere Dienstleister den Hersteller auf dem Support-Markt schlagen.

cc) Produktivitätsverlust durch Forks

Hinzu kommt, dass Forks[1] Produktivität vernichten können. Bei Closed- und Shared-Source hat der Hersteller allein die Möglichkeit, die Richtung der Entwicklung des Produktes zu bestimmen. In Open-Source-Projekten kann es bei divergierenden Meinungen über die Weiter-

1. Aufteilungen der Weiterentwicklung in zwei divergierende Richtungen.

entwicklung zu einem Fork kommen. Für jeden Zweig wird in der Regel nur ein Teil der Entwicklerkapazitäten zur Verfügung stehen. Einerseits ist dies ebenso ineffizient wie die Entwicklung eines Parallelproduktes durch einen Mitbewerber im Closed-/Shared-Source-Bereich. Andererseits kann es dazu führen, dass beide Teile durch die Auseinandersetzung das Interesse am Projekt verlieren[1]. Bei Closed-/Shared-Source ist dies weniger zu befürchten, da die Hersteller typischerweise von den Einnahmen abhängig sind und persönliche Animositäten nach aussen eher durch die Gewinnerzielungsabsicht abgeschwächt werden.

dd) Trend zu Massensoftware

Grundsätzlich ist es effizienter, wenn Software entwickelt wird, bei der eine möglichst grosse Zahl von Nutzern ohne Anpassung auskommt. Die Verbesserung von Fähigkeiten und Erfahrungen bringt in Folgeprojekten gerade eine Einsparung von 30 %[2]. Die Entwicklung von Massensoftware liegt aber nicht im Interesse der Open-Source-Hersteller, da sie mit Anpassungen Geld verdienen wollen.

ee) Zusammenfassung

Die Produktion ist nicht nachweisbar effizienter.

b) Distribution

aa) Transaktionskosten und Refinanzierung

Die Distribution von Open-Source ist ohne Inkasso möglich, z.B. direkt über die Webseite des Urhebers. Die Transaktionskosten sind sehr gering. Jedoch ist damit die Refinanzierung des Urhebers nicht sicher gestellt. Er wird dazu typischerweise Dienstleistungen oder Hardware anbieten. Besonders bezüglich Hardware dürfte das ineffizient sein, da sich dann ein Softwareentwickler mit Hardware beschäftigen muss und damit weniger spezialisiert sein kann[3]. Wenn ein Entwickler sich über einen Distributor refinanziert, so kann dies auch nicht effizienter sein, als wenn er in einer herkömmlichen Softwarefirma arbeitet.

bb) Erfolgskontrolle und Leistungsvergleich

Ein weiterer Aspekt ist die Erfolgskontrolle, die mit dem Inkasso verbunden ist. Herkömmliche Softwarehersteller wissen, wer das Pro-

1. Weil sie es nicht mehr als „ihr“ Projekt verstehen können. So fällt z.B. bei Stallmann/Raymond auf, wie egozentrisch seine Ausführungen in „Bazaar&Cathedral“ sind. Es ist schwer vorstellbar, dass er in untergeordneter Rolle weiter an einem Projekt mitarbeiten würde. So beklagte er sich häufig darüber, dass Linux im Grunde GNU/Linux heissen müsse, da Linux zu einem wesentlichen Teil auf GNU basiere.
2. Vgl. Hoch/Roeding/Purkert/Lindner S. 43.
3. Auch Firmen haben Kernkompetenzen.

gramm lizensiert hat[1]. Damit können sie Kontakt zur Zielgruppe herstellen. Sie wissen aber auch, wieviel die Software ihr wert war, können darüber den Nutzen einschätzen, in Relation zu den Wettbewerbern setzen und daraufhin ihre Effizienz erhöhen.

cc) Zusammenfassung

Die Distribution von Open-Source ist nicht erweisbar effizienter als bei herkömmlichen Modelle.

c) Zusammenfassung

Weder die Produktion noch die Distribution von Open-Source scheint bei einer Gesamtbetrachtung effizienter zu sein. Es steht zu befürchten, dass die Effizienz sich weiter verschlechtert, wenn zusätzliche gesetzliche Auflagen hinzukommen[2]. Für die fühlt sich zur Zeit niemand verantwortlich.

2. Dezentralisiert

aa) Geografisch

Dezentralisierung gibt es in zwei verschiedenen Formen: national und international. Grundsätzlich kann internationale Dezentralisierung Vorteile bieten. Beispielsweise können Tag und Nacht keine Rolle mehr spielen, da irgendwo auf der Welt immer gerade gearbeitet wird. Auch sind die Projekte nicht mehr anfällig für nationale Probleme wie Streiks. Jedoch müssen kulturelle Unterschiede überbrückt werden. Es dürfte gerade für Projekte mit nationalem Hintergrund[3] schwierig sein, diese in internationalen Teams zu lösen. Bei nationaler Dezentralisierung gibt es die Probleme zwar nicht mehr, aber der Vorteil der Arbeitszeiten fällt weg. Es bleibt die Möglichkeit, sich die qualifiziertesten Kräfte innerhalb des Landes oder international zu suchen. Herkömmliche Unternehmen nutzen diese Chance schon längst.

Ein gewichtiger Nachteil der Dezentralisierung besteht darin, dass die Wege länger werden und damit die Kommunikation erschwert wird. Zwar stehen mit Email und Newsgroups schnelle Medien zur Verfügung. Aber viele Dinge lassen sich schriftlich nur schwer klären[4].

Grundsätzlich ist die Dezentralisierung kein Alleinstellungskriterium für Open-Source. Herkömmliche Firmen entwickeln ebenfalls dezen-

1. Oder könnten es zumindest leicht in Erfahrung bringen.
2. Diese könnten z.B. in der Berücksichtigung rechtlicher Regelungen hinsichtlich Qualität und Patentansprüchen auf Software bestehen.
3. Z.B. Buchhaltungssoftware.
4. So wurde z.B. das anfangs föderal auf die Länder verteilte Projekt Fiscus zentral nach Bonn überführt, vgl. http://www.datenschutz.thueringen.de/tb4/tb4_9.htm.

tral, so weit dies Vorteile bietet.

bb) Verantwortung

Auch die Verantwortung wird bei Open-Source auf viele Personen dezentralisiert. Das könnte bezüglich der Einhaltung von Rechtsnormen ebenso problematisch sein wie bei der Rechtsdurchsetzung.

Dezentralisierung hat auch unter Sicherheitsaspekten keine eindeutigen Vorteile. So könnte z.B. eine Regierung genau so veranlassen, dass Überwachungscode in Software eingebaut wird, wie sie den Verwalter eines Open-Source-Projektes dazu bringen kann[1].

cc) Zusammenfassung

Weder die geografische noch verantwortungsmässige Dezentralisierung ist ein spezifischer Vorteil von Open-Source.

3. Zusammenfassung

Das Open-Source-Modell bietet keine eindeutigen Vorteile bezüglich der Effizienz oder Dezentralisierung gegenüber herkömmlichen Modellen der Softwareentwicklung. Zu den Vorteilen lassen sich jedenfalls auch entsprechende Nachteile konstruieren[2].

III. Gegenargumente

a) Warum nur bei Software?

Wenn Open-Source ein so gutes Geschäftsmodell ist, mag man sich die Frage stellen, wieso nicht längst alle Kinofilme kostenlos vorgeführt werden und die Kino-Betreiber nur von innovativen Dienstleistungen, z.B. Platzanweisen und Filmberatung, oder Hardwareverkauf, z.B. in Form von Popcorn, leben[3]. Oder Verlage ihre Bücher schon längst unter der Open Publication Licence[4] verbreitet haben. Die Autoren könnten auch mit Dienstleistungen Geld verdienen, z.B. durch persönliches Vortragen oder Schreiben von individuellen Werken. Die Hersteller von Musik wehren sich sogar energisch dagegen, dass man ihre Werke kostenlos über das Internet verbreitet und sehen es überhaupt nicht als förderlich für den Vertrieb von Dienstleistungen (z.B. Live-Konzerte) oder den Verkauf von Fan-Material an[5]. Die wirtschaftliche Situation der

1. Vgl. McGowan 271 am Beispiel Microsoft und Linus Torvalds.
2. Z.B. dass durch die Dezentralisierung die Einhaltung von Gesetzen schwerer zu gewährleisten ist, vgl. McGowan 271 zu DeCSS, oder unspektakulärer die Durchsetzung von Mängel- und Haftungsrechten sowie gewerblichem Rechtsschutz.
3. Der Vergleich mag auf den ersten Blick absurd wirken. Der Film wird durch das Vorführen aber ebenso wenig verschlissen wie Software. Das Kino selbst kann man durchaus mit der Investition des Software-Entwicklers in seine Hardware und Software vergleichen.
4. Vgl. http://www.opencontent.org.
5. Vgl. den Streit von RIAA und Napster, http://www.heise.de/newsticker/data/cp-09.08.01-000.

Distributoren spricht eine deutliche Sprache. Für die von Raymond beschriebene „Geschenkkultur“[1] gibt es keine erfolgreichen Beispiele. Stattdessen basiert die Wirtschaftsordnung auf der Entlohnung von Mehrwert. Wenn sich mit Moglen[2] tatsächlich bestreiten liesse, dass kommerzielle Anreize kreatives Arbeiten fördern, so müsste es dafür Belege auch in anderen Branchen geben. Bezüglich der Ausübung eines Hobbies in der Freizeit mag dies gelten, nicht jedoch, wenn die Urheber mit ihrer Tätigkeit den Lebensunterhalt sichern müssen. Denn dann werden sie, bedingt durch die Opportunitätskosten, die Priorität auf Tätigkeiten legen, bei denen kommerzielle Anreize überwiegen[3]. Die breite Masse der professionellen Entwickler wird in der Freizeit etwas besseres zu tun haben, als weiter Software zu entwickeln, um sie zu verschenken[4]. Es hat den Anschein, dass die Nutzung von Allgemeingut mit seiner Erschaffung verwechselt wird[5]. Eine Software ist als solche noch nicht als Allgemeingut vorhanden[6], sie muss erst erschaffen werden. Zentrales Anliegen des Urheberrechts ist die Ermöglichung bzw. Förderung der Erschaffung von Werken. Daher ist es irrelevant, dass Software nicht wie andere Resourcen verbraucht wird[7]. Die Aussage von Torvalds, dass gute Programmierer sowieso nicht für Geld oder Anerkennung, sondern aus Spass arbeiten, ist realitätsfern und zeugt davon, dass er entweder keine professionellen Entwickler kennt oder die Qualität ihrer Arbeit nicht richtig einschätzt[8]. Young behauptet hingegen, die überwiegende Zahl der Linux-Entwickler seien professionelle Software-Entwickler aus grösseren Software-, Ingenieurs- oder For-

1. Danach streben die Mitglieder nach Anerkennung, indem sie Geschenke verteilen, vgl. McGowan 262 bzw. Raymond S. 80 ff. Offen bleibt, wie diese Geschenke finanziert werden. Raymond verkennt, dass zwar bezüglich Speicherplatz, Bandbreite und Rechenzeit nur selten ein Mangel herrscht, aber diese auch, besonders aber die aufgewendete Arbeitszeit, finanziert werden müssen.
2. Vgl. die Darstellung von McGowan 266 sowie Moglen I.
3. Vgl. auch die historischen Überlegungen, die davon ausgingen, dass der Drucker/Verleger zumindest die Chance auf einen dem Zinsertrag gleichen Gewinn haben müsse, weil er anderenfalls sein Geld nicht in das Werk investiere.
4. Aus den selben Gründen wie z.B. Ärzte, Rechtsanwälte oder Steuerberater selten in ihrer Freizeit kostenlos zur Weltverbesserung beitragen. Sofern die Entwickler fest angestellt sind könnten sie sogar Wettbewerbsklauseln verletzen bzw. den Bestand ihres Arbeitgebers gefährden, wenn sie dort erworbenes Knowhow verwenden.
5. Vgl. McGowan 267 sowie Moglen III.
6. Ungleich beispielsweise der Atemluft.
7. Im übrigen wird sehr wohl die Refinanzierungschance des Urhebers „verbraucht“, je öfter die Software kostenlos verbreitet wird.
8. Vgl. McGowan 269 bzw. das Interview mit Linus Torvalds unter http://firstmonday.org/issues/issue3_3/torvalds/index.html. Man wende die Aussage einfach probeweise auf die oben genannten anderen Berufsgruppen an. Die These scheitert an der Erklärung, wie diese Menschen ihren Lebensunterhalt bestreiten bzw. leugnet ihre Qualifikation. Vgl. auch kritisch zur herkömmlichen Charakterisierung der Motive und einer Welt voller Schenker und Beschenkter: Sester S. 798.

schungsorganisationen[1]. Torvalds Vergleich von Open-Source mit Wissenschaft überzeugt auch nicht[2]. Zwar hat Open-Source auch positive Sekundäreffekte bezüglich einer Wohlstandserzeugung. Jedoch dürfte allgemein bekannt sein, dass Wissenschaft von staatlicher Seite finanziell unterstützt werden muss, um überlebensfähig zu sein. Das würde wiederum übertragen auf Open-Source bedeuten, dass der Staat die Entwickler subventionieren müsste, weil diese ihre Arbeit freiwillig verschenken. Das ist eine seltsame ökonomische Folge der Argumentation gegen Intellectual Property Rights[3].

b) Warum erst jetzt?

So etwas wie Open-Source wäre grundsätzlich bereits lange vorher möglich gewesen. Gerade bei Texten, Musik und Filmen könnte man heute argumentieren, dass aufgrund der vielfältigen und auch für Privatleute erschwinglichen Produktionstechnologien[4] geradezu ein Ansturm auf die Veröffentlichung im Internet hätte einsetzen müssen. Davon ist interessanterweise nichts zu bemerken[5], und das, obwohl das Schreiben eines Textes oder Aufnehmen eines einfachen Liedes der Ausbildung des Durchschnittsbürgers wohl deutlich näher liegt als die Entwicklung einer Software[6]. Die Verbreitung dieser Werke über das Internet würde nicht viel mehr Geld kosten als die Verbreitung von Software. Wäre Open-Source tatsächlich eine Art „Patentrezept“ im Zusammenhang mit digitalem Content, so müsste es in diesen Bereichen viel mehr Werke geben. Es ist nicht einsehbar, warum mehr Leute in ihrer Freizeit mühsam den Linux-Kernel nach Fehlern untersuchen sollten, als ein einfaches Lied aufzunehmen und es zur Freude anderer ins Internet zu stellen. Auch im kommerziellen Bereich ist in anderen Branchen nichts von derartigen Geschäftsmodellen zu sehen.

c) Verdacht auf andere Ursachen

Dass derartige Werke nicht auf breiter Front geschaffen werden, lässt für Open-Source nur den Schluss zu, dass hier andere Mechanismen zur Popularität geführt haben. Die erste Version der GPL stammt immerhin aus dem Jahr 1989, die Idee ist damit nicht besonders neu[7]. Besonders

1. vgl. glaubwürdig DiBona/Ockman/Stone/Young S. 122. Er zählt IBM, HP, Digital, Sun ebenso auf wie die NASA oder allgemein die Hardware-Hersteller selbst.
2. Vgl. Torvalds S. 228.
3. Vgl. Torvalds S. 204-214.
4. Z.B. Textverarbeitungssoftware, Musiksoftware, Digitalkameras.
5. Zumindest wenn man eine minimale Qualität in die Betrachtung einbezieht.
6. Die meisten Deutschen können Schreiben und Lesen, auch spielen durchaus mehr Menschen ein Instrument oder singen, als es Softwareentwickler gibt.

die Nützlichkeit für grosse Konzerne und deren finanzielle Unterstützung wird dazu beigetragen haben[1]. Daher steht zu befürchten, dass Open-Source wieder eher zu einem Nischendasein zurückkehren wird, wo sie sich einmal aus eigener Kraft am Leben erhalten muss. Dafür spricht auch, dass die Protagonisten[2] überwiegend schon 10 Jahre und länger existieren, aber erst zur Zeit des Börsenbooms entsprechende Popularität erlangten. Als Beweis der Funktionsfähigkeit des Modelles taugen sie also gerade nicht. Vielversprechender sind Modelle, bei denen Open-Source nicht aus Altruismus oder ideologischen Gründen, sondern offen als Marketinginstrument eingesetzt wird[3].

d) Höhe der Entwicklungsetats

Microsoft hat in die Entwicklung von Windows 95 etwa $ 1 Milliarde investiert[4]. IBM hat sich OS/2 rund $ 2 Milliarden kosten lassen[5]. Warum investierten diese Firmen so viel Geld, wenn sich im Open-Source-Modell viel günstiger Software entwickeln liesse? Zumindest bei Microsoft ist nicht zu erkennen, dass man die Strategie hin zu Open-Source verändert. Dass IBM nun die Entwicklung von Linux mit mehreren Milliarden Dollar[6] fördert, spricht ebenfalls nicht dafür, dass das Modell effizienter sein könnte[7]. Zudem haben vier der weltweit größten Linux-Distributoren (Caldera, Conectiva, SuSE und TurboLinux) bekannt gegeben, dass sie zur Einsparung von Entwicklungskosten die Initiative „United Linux“ gründen[8]. Das wäre wohl kaum notwendig, wenn die Entwicklungskosten aufgrund der Überlegenheit des Modells marginal wären bzw. diese Firmen mit Open-Source ein überlegenes Geschäftsmodell hätten.

e) Zersplitterung in der Unix-Historie

Open-Source hat seine Ursprünge im Unix-Bereich. Gerade dieser war dafür bekannt, dass jeder Hersteller sein eigenes Süppchen kochen

7. Vgl. den ersten Copyright-Vermerk in http://www.fsf.org/licenses/gpl.txt. Richard Stallman hatte diese Ideen bereits 1985, vgl. McGowan 261.
1. Nur 3,8 % des Umsatzes für Server-Hardware entfallen auf Linux-Server, allerdings stieg dieser Anteil im Vergleich des ersten Quartals von 2001 zu 2002 um 54,7 %. Davon profitiert Marktführer und Linux-Sponsor IBM mit einem Marktanteil von 27,8 %, vgl. http://www.heise.de/newsticker/data/anw-29.05.02-005.
2. Z.B. Linux und StarOffice.
3. Z.B. bei IBM-Hardware, oder im Softwarebereich bei MySQL oder Qt von Trolltech (dort ist die Linux-Version frei für nicht-kommerzielle Anwendungen). IBM hat allerdings selbst eine sehr unrühmliche Geschichte bezüglich Monopolen.
4. Vgl. Hoch/Roeding/Purkert/Lindner S. 39.
5. Allerdings incl. Marketingkosten, vgl. Hoch/Roeding/Purkert/Lindner S. 146.
6. Vgl. „k) weil sie gefördert wird” auf Seite 37.
7. Dann würde das Geld bei den Distributoren längst verdient und es bräuchte keine Sponsoren wie IBM, Oracle oder SAP.
8. Vgl. http://www.heise.de/newsticker/data/odi-29.05.02-000.

wollte[1]. Zum Leidwesen der Anwender, denn kaum ein Programm lief ohne Anpassungen auf einem anderen Unix. Schliesslich verliess ein Hersteller nach dem nächsten den Markt. Der Markt insgesamt wurde durch die Anpassungsprobleme immer kleiner. Ein Markt für Massensoftware konnte sich nicht ausbilden. Im Idealfall wurde das Programm im Source geliefert und musste bei der Installation neu kompiliert werden, damit es zum Betriebssystem und der Hardware passte. Im schlimmsten Fall war die Software nicht nutzbar, solange man sie nicht umschrieb oder das Betriebssystem bzw. die Hardware austauschte. Standardisierung ist der Hauptgrund, warum Microsoft mit einem technisch klar unterlegenen Betriebssystem einen solchen Zulauf erhielt[2]. Einige Stimmen behaupten auch, dass Apple mit einer noch rigideren Standardisierung für vorbildliche Benutzerfreundlichkeit und effizienteres Arbeiten gesorgt hat. Es ist zu befürchten, dass sich die genannten Probleme wiederholen[3].

f) Erfahrungen mit Standards

So lange es von der Anwenderseite keine entsprechenden Vorgaben gibt[4], spricht vieles dafür, dass die Definition von einer zentralen Stelle effizienter ist als ein demokratischer Prozess nach dem Open-Source-Prinzip. Ein Beispiel dafür könnten auch die Standards CORBA[5] und J2EE/EJB[6] sein. Während sich dem demokratischeren OMG-Standard[7] folgende CORBA-Produkte nie in der Masse durchsetzen konnten, haben J2EE-Produkte durch die zentrale Vorgabe von Sun Microsystems in wenigen Jahren eine grosse Verbreitung erlangt[8]. Der SQL-Stan-

1. Es wird etwa 40 verschiedene Unix-Varianten gegeben haben, vgl. auch die Darstellung bei Raymond S. 12-15, der weniger aufzählt (nur US-Markt?). Einige sprechen von „Balkanisierung“, vgl. DiBona/Ockman/Stone/Young S. 120-123.
2. Rein technisch war z.B. MS-DOS nicht mal ein Betriebssystem im Unix-Sinne, da es die Resourcen nicht vollständig zentral verwaltet hat, vgl. Raymond S. 14 f.
3. Bzw. sich schon wiederholt haben, vgl. auch die „United-Linux“-Initiative, http://www.heise.de/newsticker/data/odi-30.05.02-000. Wenn die Distributoren als potentielle Wettbewerber sich schon zusammenschliessen, deutet das darauf hin, dass ineffizient gearbeitet wurde und es enorme Einsparpotentiale gibt.
4. Das wäre wohl eines der effizientesten Modelle. Ein Verband der Benutzer würde verbindliche Standards aufstellen und nur Software lizensieren, die sich an diese Standards hält (z.B. bezüglich Dateiformaten). Da die Benutzer bislang keinerlei Interessenvertretung haben, liefern sie sich den Herstellern aus. Sobald z.B. Microsoft eine neue Version einer Textverarbeitung heraus bringt, lizensieren Millionen von Nutzern diese nur, weil sie sonst Dateien ihrer Geschäftspartner nicht mehr lesen könnten. Die fachlichen Anforderungen an eine Textverarbeitung haben sich im wesentlichen seit dem Erscheinen von Word 1.0 am 29.09.1983 nicht geändert, das steht im krassen Gegensatz zur Anzahl der heute noch enthaltenen Fehler, vgl. zum Datum http://www.microsoft.com/msft/download/keyevents.doc oder zum stabileren TeX: http://www.tug.org/whatis.html.
5. Vgl. http://www.omg.org/corba.
6. Vgl. http://java.sun.com/j2ee/overview2.html.
7. Vgl. http://www.omg.org/news/about.
8. Allerdings ist auch hier vieles im Standard so unverbindlich definiert, dass die Produkte zur Zeit alles andere als beliebig austauschbar sind.

dard[1] wird selbst von den Marktführern nur in der Minimalversion unterstützt, da diese gerade kein Interesse an der Austauschbarkeit ihrer Produkte haben. Angesichts dieser Beispiele erscheint es fragwürdig, warum Open-Source ausgerechnet eine Ausnahme bilden sollte. Wenn alles regelnde Standards befolgt werden, können sich die Distributoren nur schwer differenzieren. Die Wechselkosten der Nutzer gingen dann gegen Null, so dass der Preis in den Vordergrund rückt, da die Produkte austauschbar sind. Arbeiten nun alle Distributoren zusammen[2], z.B. indem nur noch einer von ihnen entwickelt, entfallen ferner wichtige Aspekte des Open-Source-Modells: Der Quelltext wird so von weniger Personen und damit weniger unterschiedlichen Ansätzen entwickelt.

g) Zusammenfassung

Die Beschränkung auf Software, der Zeitpunkt und die Entwicklungsetats etablierter Firmen legen die Vermutung nahe, dass andere Ursachen für den Erfolg von Open-Source verantwortlich sind, nicht aber eine etwaige Überlegenheit des Open-Source-Modells. Auch die Historie von Unix und Erfahrungen mit Standards lassen nicht erwarten, dass eine dezentrale Entwicklung relevante Vorteile bringt.

IV. Open-Source als Korrektiv zu anderen Modellen

Einige Argumente sprechen dafür, dass Open-Source zwar nicht als alleiniges Modell taugt, aber ein gutes Korrektiv zu anderen Modellen ist.

1. Ermöglicht spontane Innovationen

Neben zahlreichen Nachahmungen bekannter Programme[3] gibt es im Open-Source-Umfeld immer wieder neue Ideen. Je höher der Aufwand ist, der neben der reinen Softwareentwicklung betrieben werden muss[4], desto mehr Kapazität geht dafür verloren. Die Wahrscheinlichkeit steigt, dass innovative Ideen überhaupt nicht umgesetzt werden, weil dem potentiellen Urheber der Aufwand bzw. das Risiko zu hoch ist.

a) Open-Source-Privileg

Open-Source ist ein hervorragendes Experimentierfeld. Als solches sollte es im Interesse der Innovationsförderung erhalten bleiben. So lange Software vom Urheber zur freien und kostenlosen Verfügung bereit

1. Vgl. Schneider/Werner S. 491.
2. Oder zumindest die grössten, siehe die „United Linux“-Initiative, vgl. http://www.heise.de/newsticker/data/odi-29.05.02-000; ein Grossteil der Entwicklung soll nun nur noch in Nürnberg stattfinden, vgl. http://www.heise.de/newsticker/data/odi-30.05.02-000.
3. Vgl. sehr kritisch zur Innovationsfähigkeit von Open Source: http://www.computerwoche.de/index.cfm?pageid=254&artid=17392.
4. Z.B. für Patentrecherchen, Lizensierungsfragen, rechtliche Absicherung.

gestellt wird, sollten andere staatlich gewährte Monopole, insbesondere (Software-)Patente, zurückstehen. Grundsätzlich existiert zwar ein „Experimentierprivileg“ in der Form, dass Private jederzeit patentierte Techniken einsetzen dürfen, ohne lizenzpflichtig zu sein. Jedoch endet dieses unmittelbar bei einer Verbreitung des Werkes. Bei Open-Source ist der interessante Punkt jedoch gerade der Austausch und das Zusammenwirken vieler Urheber, mit der Chance, dass das Gesamte mehr wird als die Summe der Einzelteile. Das spricht für ein „Open-Source-Privileg“. Es könnte so ausgestaltet sein, dass Open-Source-Software grundsätzlich keinen Restriktionen durch das Patentrecht unterliegt.

b) Missbrauch möglich

Die vorsätzliche Schädigung von Patentinhabern, z.B. durch ihre (ggf. ehemaligen) Mitarbeiter, könnte zum Problem werden. Diese könnten bei Existenz eines Open-Source-Privilegs einfach den Anspruchsinhalt in Open-Source-Software münzen und würden damit dem Patentinhaber eine Möglichkeit zur Refinanzierung nehmen[1]. Er wäre dann gezwungen, sich dem Open-Source-Modell zu stellen. Gerade für aufwendige Erfindungen wäre die Refinanzierung zweifelhaft, denn der Quelltext kann vermutlich sehr schnell entwickelt werden, wenn erst einmal die ihm zugrunde liegende Idee durch das Patent offenbart ist.

Jedoch muss man die zusätzliche Anforderung der kostenfreien Abgabe beachten. Der Hersteller des Open-Source-Werkes kann also zumindest durch das Werk selbst keinerlei Refinanzierung vornehmen. Wenn jemand kostenlos ein darauf basierendes Werk erstellen kann, und seine Kosten tragen kann, so müsste das aber dem Patentinhaber durch seinen Vorsprung erst recht gelingen.

Ein Aspekt, unter dem Softwarepatente diskutiert werden, ist der mögliche Schutz des „kleinen“ Erfinders vor übermächtigen Konkurrenten. Diese könnten ein Open-Source-Privileg nutzen, den Patentinhaber schädigen, wären aber zugleich gar nicht auf eine Refinanzierung auf Basis des betreffenden Anspruchs angewiesen. Dagegen ist einzuwenden, dass es auch so schon genügend Möglichkeiten für diese Unternehmen gibt, dem Patentinhaber das Leben schwer zu machen[2].

c) Zusammenfassung

Die Einräumung eines Open-Source-Privilegs sollte näher geprüft wer-

1. Die Möglichkeit aus der Lizensierung des Patentes Einnahmen zu erzielen fällt weg, wenn jeder kostenlos entsprechende Software erhalten kann.
2. Z.B. durch Sperrpatente oder prozessuales Vorgehen.

den. Sie könnte Innovationen fördern, weil sie Open-Source-Entwickler von der Verpflichtung frei stellt, erst umfassend rechtliche Ansprüche zu prüfen, bevor sie Software erstellen. Ein solches Privileg könnte zudem ein Korrektiv zu Softwarepatenten darstellen[1].

2. Verhindert Blockade grosser Hersteller

Frei nach Lawrence Lessig ist festzustellen, dass Open-Source dafür sorgt, dass Märkte aufgerüttelt werden und innovativ bleiben[2]. Man verhindert damit eine Blockadepolitik der grossen, etablierten Anbieter, die lieber mit alten Technologien Geld verdienen, so lange sie nicht unbedingt auf neue Technologien umsteigen müssen. Ein Spezialfall ist das „Innovator's Dillemma"[3]. Danach sind Firmen nur so lange innovativ, wie sie keine sichere Kundenbasis haben. Das liegt daran, dass es ihnen ratsamer erscheint, existierende Produkte zu verbessern, statt völlig neue Märkte aufzubauen[4]. So ist der Wettstreit mehrerer Anbieter dem Fortschritt förderlich. Es hat Vorteile, wenn die Interessen sowie die Struktur der Anbieter sehr verschieden sind. Das ist bei herkömmlichen Softwarefirmen und Open-Source-Produzenten der Fall. So ergibt sich auch ein Wettstreit der Modelle. Es kann vorteilhaft sein, Urhebern über das Open-Source-Modell die ungefilterte, nicht durch Verleger im weitesten Sinne nivellierte Darstellung ihrer Werke zu ermöglichen[5]. Die Chance dürfte gross sein, dass auf diesem Wege Software zustande kommt, die sonst den Kunden nie erreicht hätte, im positiven wie im negativen Sinne[6].

3. Dokumentiert den Stand der Technik

Normalerweise hat die Öffentlichkeit keinen Zugang zu den Quelltexten. Diese erhalten allenfalls die Kunden von Individualsoftware oder bei Abschluss spezieller Non-Disclosure-Agreements. Dadurch ist es schwer, den Stand der Technik eindeutig zu bestimmen. Bei Open-Source jedoch wird der Quelltext veröffentlicht, so dass er für Dritte

1. Insbesondere gegen Trivialpatente.
2. Lessig, S. 222, zur Regulierung von Funkfrequenzen; Plädoyer zum Offenhalten eines bestimmten Funkbandes für unlizensierte Nutzung durch jedermann, damit neue Firmen einfachen Zugang zum Markt erhalten. Quasi als Spielwiese.
3. Vgl. Christensen S. 26
4. Weil dies auch mit Risiken verbunden ist, die man (da bereits erfolgreich) nicht unbedingt eingehen muss, vgl. Christensen S. 97.
5. Vgl. Litman S. 102 f. zur Verfügbarkeit von Werken im Internet, die so nicht kommerziell erhältlich sind. Auch im Softwarebereich ist häufig erkennbar, dass Entwickler eine andere Meinung über die Gestaltung eines Produktes haben, als eine Marketingabteilung (die z.B. Features nicht sofort einbauen möchte, sondern über mehrere Versionen verteilen, um grössere Einnahmen zu sichern). Erstaunlicherweise ist diese häufig sogar näher an den Bedürfnissen der Kunden orientiert.
6. Vgl. Litman S. 108, die von einer völlig neuen Klasse von Werken spricht, gerade weil diese nicht den herkömmlichen Marktzwängen unterliegen.

ohne Einschränkung durch Lizenzbedingungen einsehbar ist. Damit übernimmt Open-Source einen Teil der Funktionen, die das Patentwesen erfüllen sollte. Der Stand der Technik ist wichtig, um „best practices“[1] zu dokumentieren und minimale Qualitätsanforderungen zu bestimmen[2].

4. Verschafft der Allgemeinheit günstige Werke

Die Allgemeinheit erhält durch Open-Source die Möglichkeit einer günstigen Werknutzung. Ferner erhält sie zumindest Einblick in die Quelltexte der Software. Dies kann bei entsprechender Vorbildung zur Erhöhung des Bildungsniveaus dienen[3]. Zudem ist bei entsprechendem Arbeitseinsatz auch eine Fehlerbeseitigung und Fortentwicklung möglich. Open-Source macht Infrastruktur-Software[4] einer breiten Masse zugänglich und fördert den Verzicht auf Urheberrechtsverletzungen.

5. Zusammenfassung

Open-Source ermöglicht spontane Innovationen ohne grosse Markteintrittshürden. Eine erfolgreiche Blockadepolitik grosser Hersteller bei Innovationen wird durch Open-Source verhindert. Der Stand der Technik wird dokumentiert und die Allgemeinheit erhält günstige Werke im Bereich der Software-Infrastruktur.

V. Zusammenfassung

Open-Source taugt zwar nicht als alleiniges Modell für die Erstellung und Verbreitung von Software. Open-Source sollte aber vor Behinderungen, z.B. durch Softwarepatente, geschützt werden, damit Innovationen erleichtert werden.

E. ALTERNATIVEN

Neben Open-Source[5] gibt es Alternativen, mit denen die Urheber- und Verbraucherinteressen geschützt werden können.

I. Aufhebung des Urheberrechts für Software

Das entspräche im wesentlichen dem Stand vor den Urheberrechtsreformen. Bezüglich DRM sind zwei Situationen zu unterscheiden.

1. Bei mehreren möglichen Vorgehensweisen sollte sich die beste durchsetzen.
2. Mangels expliziter Vereinbarungen gilt bei Werkverträgen die mittlere Art und Güte als geschuldet. Diese ist nach dem Stand der Technik bestimmbar.
3. Z.B. durch Analyse des Quelltextes im Rahmen von Fortbildungsmassnahmen.
4. Z.B. Betriebssysteme, Office-Systeme, Webserver, EJB-Container.
5. Sandl sieht in Open-Source sogar Lösungsansätze, die für den weiteren Verlauf des Informationszeitalters entscheidend sind (S. 346). Im Kern nennt er Sicherheit, Vertrauensförderung und „offene“ Schnittstellen.

1. DRM verboten und nicht geschützt

Digital Rights Management-Systeme könnten verboten sein[1] bzw. ihre Umgehung keinen Sanktionen unterliegen.

a) Ungehemmtes Kopieren

In diesem Falle wäre die Situation ähnlich wie in der jetzigen Praxis des mangelhaft implementierten Urheberrechtsschutzes. Darüber hinaus würden aber auch die letzten Hemmnisse fallen, da das Urheberrecht bislang zumindest Firmen davon abhält, allzu dreist Software illegal zu kopieren. Die Situation käme wirtschaftlich für die Urheber der zwangsweisen Verwendung von Open-Source-Lizenzen gleich.

b) Interessen stark einseitig berücksichtigt

Die Interessen der Verbraucher wären zwar umfänglich dadurch berücksichtigt. Durch die allgemeine Verfügbarkeit der Software stiege das Bildungsniveau. Weil Software an sich nichts kostet, könnten sich grosse Teile der Bevölkerung legal in diese Programme einarbeiten. Damit stünde dem Arbeitsmarkt ein Heer gut ausgebildeter Programmnutzer zur Verfügung. Aber auch die Urheber können davon profitieren. Sie müssen zur Erstellung ihrer eigenen Werke geringere Kosten aufbringen, da sie selbst keine Lizenzabgaben mehr zahlen müssen. Dadurch sind die Startinvestitionen geringer und es wird eher der Break-Even-Point erreicht, wodurch die Hemmschwelle sinken dürfte. Das Innovationstempo dürfte sich so zunächst erhöhen. Dennoch wäre höchst fraglich, ob bestimmte Arten von Software überhaupt noch entwickelt würden, da ihre Refinanzierung auch im Open-Source-Modell zweifelhaft erscheint. Insbesondere Software für spezielle Anwendungsbereiche und aufwendig zu erstellende Programme mit hohem Verbrauchernutzen würden kaum noch entwickelt, da ihre Refinanzierung eher unwahrscheinlich ist. Die Berücksichtigung der Interessen würde also stark zu Gunsten der Verbraucher verschoben.

c) Kaum vereinbar mit dem Eigentumsschutz des Grundgesetzes

Es bestehen starke Zweifel, ob dies mit dem grundgesetzlich garantierten Eigentumsschutz gemäss Art. 14 GG verträglich ist. Immerhin könnte es einer Enteignung gleich kommen, wenn der Urheber nicht mehr über sein Werk bestimmen kann.

d) Ausweichlösungen

Die Urheber könnten auf andere Verwertungsarten umsteigen, z.B. Ap-

1. Z.B. um das Recht auf Privatkopie zu erhalten (gilt nicht für Software).

plication Service Providing (ASP). Sie könnten ihre Werke, bis auf die Schnittstellen, geheim halten, indem sie die Funktionalität auf Server verlagern, die z.B. über das Internet erreichbar sind[1]. Lediglich die Ein- und Ausgabe verbliebe auf dem Rechner des Nutzers, die Verarbeitung erfolgt komplett auf dem zentralen Server. Damit wäre der bislang geschützte Teil der Software nicht einsehbar oder kopierbar für den Nutzer. Der Urheber hätte so die Möglichkeit, die Nutzung zu überwachen und abzurechnen. Er müsste vor allem sein Werk und ggf. Geschäftsgeheimnisse nicht aus der Hand geben[2].

Der Nutzer müsste keine Aktualisierungen mehr durchführen und ggf. nur das bezahlen, was er wirklich nutzt. Für berechnungsintensive Programme können so zentrale Rechenkapazitäten effizienter ausgelastet werden. Der Nutzer erspart sich die Anschaffung teurer Hardware.

Problematisch wird allgemein eingeschätzt, dass der Urheber dadurch Zugriff auf die Daten erhält[3]. Auch ist die Kommunikation verhältnismässig leicht abhörbar. Ein wichtiges Kriterium dürfte auch sein, dass der zentrale Server dem Zugriff des Nutzers entzogen ist, und es historisch eher schlechte Erfahrungen mit diesem Modell gibt, da die Verfügbarkeit solcher Zentralrechner häufig zu wünschen übrig ließ[4]. Es kommen technische Probleme z.B. bei der Übertragung von Druckdaten hinzu[5].

e) Zusammenfassung

Insgesamt ist eine Aufhebung des urheberrechtlichen Schutzes für Software ohne DRM nicht sinnvoll. Application Service Providing könnte nur in wenigen Bereichen als Problemlösung dienen.

2. DRM erlaubt und geschützt

Man könnte allerdings auch DRM erlauben und seine Umgehung unter Strafe stellen. In diesem Fall könnten sich die Urheber gleichsam selbst helfen, indem sie es den Verbrauchern zumindest stark erschweren, die Software zu kopieren.

1. Ggf. nur für bestimmte Benutzer nach vorheriger Anmeldung oder über andere Datennetze, um Internet-typischen Problemen wie denial-of-service-Angriffen oder Sicherheitsproblemen aus dem Wege zu gehen.
2. Vgl. die Angebote im juristischen Bereich: Juris, http://www.juris.de und Beck Online, http://www.beck.de.
3. Das hat auch eine datenschutzrechtliche Komponente, so wird es kritisch betrachtet, wenn z.B. die Polizei, Steuerbehörden oder Banken ihre Rechenzentren outsourcen würden und dadurch ggf. unbefugte Dritte Einsicht nehmen könnten.
4. Z.B. Leitung überlastet/defekt, Hardwarefehler, single point of failure.
5. Vgl. „Zu viele Stolpersteine für ASP“, COMPUTERWOCHE Nr. 45 vom 9.11.2001, http://www.computerwoche.de/index.cfm?pageid=267&type=Artikel-Detail&id=80105308&cfid=4796566&cftoken=31071324&nr=10.

a) Einnahmen aus Primärverwertung

Obwohl das Urheberrecht keine Property Rights mehr zur Verfügung stellt, würden die Urheber Gewinne aus der Primärverwertung erzielen können, da sie ihre Werke mit DRM-Systemen schützen. Der Nutzer bekommt also nur bei Nutzung eines DRM-Systems die Möglichkeit zur Benutzung des Werkes. Dafür kann der Urheber abrechnen.

b) Beschränkung durch DRM-Systeme

Problematisch daran ist, dass DRM-Systeme keine absolute Sicherheit bieten. Jedes DRM-System ist umgehbar, das ist lediglich eine Frage des Aufwandes[1]. Aber das heutige Urheberrecht ist noch viel schlechter implementiert, als es durchschnittliche DRM-Systeme in Zukunft sein könnten. Für den Grossteil der Nutzer, die keinen Zugriff auf Spezialhardware haben, würde das Kopieren erschwert oder ganz verhindert. Es laufen bereits Vorbereitungen, eine Version des Betriebssystems Windows so auszugestalten, dass durch ein integriertes DRM-System ab dem Startvorgang des Rechners kontrolliert wird, ob die DRM-seitige Berechtigung zur Nutzung einer Datei besteht[2]. Das umfasst dann also nicht nur Content wie Musikdateien, sondern auch schon Treiber und normale Anwendungsprogramme. Sofern auf dem Markt nur noch Hardware und Betriebssysteme erhältlich wären, die DRM unterstützen, ist davon auszugehen, dass die Verfügbarkeit illegaler Kopien drastisch zurück gehen würde[3].

c) Privatkopie nicht sicher gestellt

Ein umfassender Schutz über DRM könnte dazu führen, dass dem Verbraucher kein Recht auf eine Privatkopie zur Weitergabe im Freundeskreis mehr gewährt wird[4]. Eine Gewährung der Privatkopie könnte zur Aushöhlung der DRM-Systeme insgesamt genutzt werden. Sofern die Wirksamkeit der DRM-Systeme gewährleistet wäre, würden sich allerdings auch keine illegalen Kopien weitergeben lassen. Das ist zwar genau das Ziel dieser Systeme, kann aber auf der anderen Seite dazu führen, dass das Bildungsniveau der Bevölkerung hinsichtlich Software deutlich sinkt. Schliesslich könnte man sich nur noch in Software einarbeiten, sofern man eine entsprechende Lizenz dafür besitzt. Unter Effi-

1. Das folgt daraus, dass diese Systeme in der Regel auf kryptografischen Verfahren basieren.
2. Vgl. das MS-Patent: http://www.heise.de/newsticker/data/cgl-14.12.01-000.
3. Weil die Erstellung entweder sehr aufwendig wäre oder man den Kopierer leicht ausfindig machen könnte.
4. Dieses existiert für Software sowieso nicht.

zienzgesichtspunkten ist das jedoch sinnvoll. Denn zur Zeit werden rechtskonforme Nutzer bestraft, da sie Geld für die Lizenz zahlen, wobei die illegalen Nutzer sich genauso einarbeiten können. Dieser Mitteleinsatz führt nicht zu einem spezifischen Nutzen und ist ineffizient.

d) Zusammenfassung

DRM-Systeme sollten durch entsprechende Gesetze gefördert werden. Denkbar wäre sogar die verpflichtende Einführung.

3. Zusammenfassung

Eine Aufhebung des urheberrechtlichen Schutzes für Software ist zur Zeit nicht empfehlenswert. Sobald überwiegend DRM-Systeme im Markt verfügbar sind, könnte der Bedarf nach herkömmlichem urheberrechtlichem Schutz wegfallen. Dafür entstünde Bedarf am Schutz gegen Umgehung und für Verbreitung der DRM-Systeme.

II. Pflicht zur Veröffentlichung der Schnittstellen

1. Problem

a) Erschwerter Produktwechsel für den Verbraucher

Der Nutzer hat ggf. grosse Datenmengen mit dem Produkt erstellt, die bei einem Wechsel wertlos würden, da andere Programme sie nicht verarbeiten können[1]. Alternativ würden grosse Kosten auf ihn zu kommen, um die Daten in das Format des neuen Programmes zu konvertieren.

b) Markteintrittshindernis

Für die Mitbewerber ergibt sich ein Markteintrittshindernis daraus, dass die Anwender nur schwerlich von einer etablierten Software auf ein neues Produkt umsteigen können, sei dieses Produkt auch noch so gut[2]. Für die Entwickler von Drittprodukten, die Schnittstellen zu dem Produkt nutzen wollen, ergibt sich der Zwang, Verträge mit dem Produkthersteller eingehen zu müssen, um die benötigten Unterlagen zu erhalten oder selbst die Schnittstellen mühsam erforschen zu müssen. Der Produkthersteller wird häufig kein Interesse daran haben, derartige Drittprodukte zu fördern, sofern er nicht sicher sein kann[3], dass die Treue des Drittherstellers unzweifelhaft ist. Das führt zur Monokultur.

c) Zusammenfassung

Das Hauptproblem der Nutzer und Mitbewerber lautet: Die Schnittstellen zu einem Produkt sind nicht ausreichend spezifiziert und verfügbar.

1. Vgl. „c) Investitionssicherung" auf Seite 26.
2. Ein Problem, das z.B. StarOffice fast 15 Jahre begleitete.
3. Z.B. über Knebelverträge, NDAs.

2. Regelungsinhalt

Aus diesem Grund könnte der Hersteller einer Software zur Veröffentlichung der Schnittstellen verpflichtet werden. Dem berechtigten Nutzer könnte ein Rechtsanspruch darauf eingeräumt werden, die Schnittstellen kostenlos zu erhalten sowie kostenfreien Support vom Hersteller zu erhalten, bis eine Konvertierung möglich ist. Dabei spielt es keine Rolle, unter welchen Lizenzbedingungen die Software selbst veröffentlicht wird. Diese Regelung könnte man notfalls auf Hersteller mit einem bestimmten Marktanteil einschränken, z.B. 10 Prozent.

Im Detail müssten sämtliche persistenten Daten[1] des Programmes über diese Schnittstelle verlustfrei lesbar sein. Die Schnittstelle müsste für einen sachverständigen Dritten in angemessener Zeit genutzt werden können. Die Schnittstelle ist auf Lesezugriffe beschränkt, da es primär darum geht, den Produktwechsel zu erleichtern, um den Wettbewerb zu erhalten. Sofern man den Investitionsschutz erhöhen und die Wiederverwendung durch Dritthersteller fördern möchte, könnte man auch die Schreibbarkeit über diese Schnittstelle fordern. Dies würde auch die Überprüfung der Vollständigkeit der Schnittstellenbeschreibung erleichtern.

3. Kosten

a) Urheber

Es fallen durch diese Regelung zusätzliche Kosten bei den Urhebern an. Sie müssen die Schnittstelle dokumentieren. Ferner müssen sie kostenlosen Support liefern. Durch den Rechtsanspruch auf Support wird gewährleistet, dass die Hersteller aus eigenem Interesse die Dokumentation so vollständig wie möglich gestalten und nicht versuchen, sie möglichst unverständlich zu fassen. Der Aufwand für die Urheber ist zumutbar. Sie müssen im Zweifel lediglich ihren dokumentierten Persistence-Layer[2] veröffentlichen. Monopole lassen sich hingegen nicht mehr so leicht aufbauen wie bisher.

b) Nutzer

Die Kosten werden von den Urhebern auf die Kunden umgelegt. Gerade bei verbreiteten Programmen mit grosser Nutzeranzahl dürfte die Belastung kaum feststellbar sein.

1. Also alle Daten, die ein Aus- und Einschalten des Rechners überdauern, beispielsweise in Datenbanken oder auf der lokalen Festplatte.
2. Die Schicht der Software, die für die dauerhafte Speicherung in Datenbanken oder auf die Festplatte zuständig ist.

c) Zusammenfassung

Die Kosten der Regelung führen zu sehr geringen Mehrbelastungen.

4. Nutzen

a) Urheber

Die Urheber profitieren ihrerseits, wenn sie Schnittstellen zu anderen Programmen einbauen möchten. Sie können mit innovativen Produkten an den Konsumenten herantreten und ihm jederzeit die Migration der Datenbasis anbieten, da sie Zugang zu den Datenformaten haben. Sie haben die Möglichkeit auf den bestehenden Daten aufzubauen und die Altsysteme zu ersetzen. Dadurch bleiben die Märkte in Bewegung.

b) Nutzer

Die Nutzer erhalten einen besseren Investitionsschutz und grössere Freiheit bei der Suche nach Alternativanbietern. Die Nutzer sind nicht länger zukünftigen Lizenz- und Schnittstellen- bzw. Dateiformatänderungen eines Anbieters ausgeliefert[1].

c) Zusammenfassung

Die Regelung dient den Interessen der Urheber und Nutzer.

5. Zusammenfassung

Der volkswirtschaftliche Nutzen dürfte bei weitem überwiegen. Die Nutzer können relativ einfach von qualitativ minderwertigen Programmen auf bessere Programme umsteigen. Andere Urheber können solche Programme erstellen, wobei sie die Daten der Altprogramme einfacher übernehmen können. Auch können sie auf bestehender Software aufbauen, ohne mit deren Hersteller spezielle Verträge schliessen zu müssen. Alles in allem erhöht sich die Effizienz, da sich unerwünschte Monopole und Abhängigkeiten zu Herstellern vermeiden lassen.

III. Schutz vor unauthorisierten Aktionen von Software

1. Open-Source-Einsatz wegen Sicherheitsbedenken

Ein Schwerpunkt des Open-Source-Einsatzes liegt im Sicherheitsbereich[2]. Dazu kann man mehrere Problemfelder betrachten.

a) Unauthorisierte Zugriffe auf persistente Daten

Programme legen ihre Daten üblicherweise auf der Festplatte ab. Bei weit verbreiteten Betriebssystemen wie Microsoft Windows gibt es kein Konzept, das Programmen verbietet, auch auf „fremde“ Festplat-

1. Dem Programm Word sagt man nach, es könne Dateien seiner eigenen Vorgängerversionen nicht einlesen. Das könnte aus Marketinggesichtspunkten nützlich sein, um die Nutzer der alten Versionen zu einem schnellen Umstieg zu bringen.
2. Beispielsweise empfiehlt das BSI Open-Source, vgl. http://www.bsi.de.

tendaten zuzugreifen[1]. Vielmehr gilt dort, dass ein Programm, sobald es erst einmal gestartet wurde, auf dem lokalen Rechner alles darf (inklusive Formatieren der Festplatte), da sich Rechte nicht pro Programm, sondern nur pro Benutzer einrichten lassen[2]. Das nutzen insbesondere sogenannte Viren, Würmer und trojanische Pferde[3] aus. Aber auch Excel XP ermöglicht unbefugten Dritten den Zugriff auf beliebige Daten[4]. Es gibt durchaus Alternativmodelle, diese sind aber nicht verbreitet[5].

b) Unauthorisierte Kommunikation über Datennetze

Darunter fällt beispielsweise das Versenden und Abrufen von Informationen an bzw. von fremden Rechnern[6]. Die Kommunikation ist in der Regel für den Benutzer nicht wahrnehmbar[7]. Sobald der Computer mit einem anderen verbunden ist, hat der Benutzer keine Gewähr mehr dafür, welche Informationen wirklich ausgetauscht werden[8]. Er ist einzig auf das Vertrauen gegenüber den Herstellern der installierten Software angewiesen. Problematisch ist das insbesondere dann, wenn Software sich selbst installiert hat, bzw. von anderer Software nachgeladen wurde[9]. Das Problem verschärft sich mit der steigenden Verbreitung von drahtlosen Netzen[10].

c) Unauthorisierte Zuordnung von Eingaben

Unter Windows NT passiert es häufig, dass der Benutzer für das Fenster mit dem Fokus eine Taste drückt, im selben Moment aber überraschend ein anderes Fenster den Fokus bekommt und dadurch die Eingabe an das falsche Fenster geleitet wird[11]. Dadurch gibt der Benutzer eine

1. Mit „fremd“ ist gemeint, dass z.B. ein Textverarbeitungsprogramm nicht nur auf Textdateien zugreifen darf, sondern auf jedwede Datei auf dem Rechner.
2. Daher hilft z.B. auch eine Umstellung auf das Dateisystem NTFS nicht.
3. Vgl. http://www.symantec.com/region/de/avcenter/vwutp.html.
4. Vgl. http://www.heise.de/newsticker/data/db-27.05.02-000.
5. Vgl. die „Jails“ unter FreeBSD, vgl. http://www.heise.de/ix/artikel/2002/03/138, oder die Sandbox von Java, vgl. http://java.sun.com/products/jdk/1.2/docs/guide/security/spec/security-spec.doc1.html.
6. Mitprotokollieren der Ein- und Ausgaben (z.B. Back Orifice, vgl. http://www.nwinternet.com/~pchelp/bo/bo.html), Ausdrucken auf fremdem Rechner/Drucker, Übertragen von Festplattendaten über Netzwerkverbindungen.
7. So lange nicht irgendwo eine zwischengeschaltete LED blinkt, merkt der Benutzer nicht, dass über ein Kabel Daten ausgetauscht werden, da es sich im Betrieb optisch nicht vom „Ruhezustand“ unterscheiden lässt.
8. Man mag den Benutzer darauf verweisen, seinen Computer eben nicht mit anderen Computern zu verbinden. Jedoch existiert das Problem prinzipiell selbst dann, wenn Daten per Diskette ausgetauscht werden. Auch da weiss der Benutzer nicht, was die Disketten enthalten.
9. Z.B. im Rahmen eines Updates oder beim Betrachten von Webseiten durch Fehler beim Interpretieren der verwendeten Skriptsprachen.
10. Vgl. zur Abhörbarkeit: http://www.heise.de/newsticker/data/ea-09.08.01-000.
11. Von sog. 0190-Dialern ist zudem bekannt, dass sie Knöpfe falsch beschriften („OK“ und „Abbrechen“ vertauscht), um eine Bestätigung des Benutzers einzuholen, die dieser sonst nie abgegeben hätte. Aber auch folgendes Beispiel ist problematisch: Jemand tippt einen Text. Plötzlich wird ein Dialog eingeblendet „Wollen Sie wirklich xyz?“, der Benutzer konnte dies nicht so schnell bemerken und hat schon im Rahmen der Texteingabe die Return-Taste gedrückt.

„Willenserklärung“ ab, die er überhaupt nicht abgeben wollte. Das ist ein verbreiteter Mangel unter Ergonomiegesichtspunkten. Es ist umso bedenklicher, da mittlerweile Willenserklärungen von erheblicher Tragweite per Computer abgegeben werden können[1]. Einen Ansatz zur Lösung des Problems in einem beschränkten Bereich stellt § 312e I 1 BGB dar, der von Unternehmern die Möglichkeit zur Korrektur von Eingabefehlern, z.B. bei elektronischen Bestellungen, fordert[2].

d) Schaden durch zugekaufte Komponenten

Entwickler, die ihrerseits Komponenten und Bibliotheken verwenden, riskieren, dass diese schädigenden oder zumindest missbrauchbaren Code enthalten, der nicht durch das darunter liegende System abgefangen wird[3]. Die Verwendung der Komponenten erfolgt in der Regel, weil das Knowhow zur Eigenentwicklung selbst nicht vorhanden ist oder man diese aus Kapazitätsgründen nicht durchführen kann. Die Entwickler müssen sich also darauf verlassen können, dass die Komponenten keine Schadfunktionen ermöglichen. Das gilt auch für das Betriebssystem oder diesem beigefügte Programme[4].

e) Analyse der Sicherheitsbedenken

Dabei wird klar, dass der Verwender längst nicht mehr wissen kann, was die Software unter seinem Namen tut[5]. Das kann zu einer Schädigung der Nutzer führen[6]. Gegen diese gibt es – bei Standardsoftware – bislang kein wirksames rechtliches Mittel[7]. Die Anbieter von Software werden sich regelmässig darauf zurückziehen, dass sie keine entsprechenden Zusicherungen gemacht haben. Problematisch ist vor allem,

1. Z.B. bei Internet-Auktionen, beachte auch die digitale Signatur, http://www.bmwi.de/Homepage/Presseforum/Pressemitteilungen/2001/1522prm2.jsp.
2. Vgl. Günther S. 94 zu Problemen bei Internet-Auktionen.
3. Z.B. die Kompressionsbibliothek zlib, die in vielen Unix-Programmen enthalten ist, vgl. http://www.heise.de/newsticker/data/pab-12.03.02-000.
4. Vgl. zum Einfluss von Scientology auf das Defragmentierungsprogramm von Windows 2000: http://www.heise.de/ct/99/25/058.
5. Deutlicher wird es, wenn man sich die Berichte über sogenannte Easter-Eggs ansieht, z.B. den in Excel 97 versteckten Flugsimulator, vgl. http://www.heise.de/newsticker/data/db-31.03.02-000 oder http://www.eeggs.com. Entweder hat dabei Microsofts Qualitätssicherung völlig versagt oder man täuscht die Kunden, die lediglich ein Tabellenkalkulationsprogramm installieren wollten. Vor allem stellt sich die Frage, welche Funktionen *noch* in derartigen Programmen stecken.
6. In der Regel durch Weitergabe ausgespähter Daten oder Manipulation von Daten.
7. Es ist kein Prozess z.B. gegen Microsoft bekannt, bei dem der Hersteller in solch einem Fall zu Schadenersatz gem. § 280 ff. BGB verurteilt worden wäre. Dazu wäre ihm eine Pflichtverletzung nachzuweisen. Das dürfte regelmässig an der mangelnden Spezifikation scheitern, was ein Betriebssystem leisten muss (vgl. § 434 I S. 2 und 3). Auch hilft der Rücktritt gemäss § 437 BGB nicht weiter: Selbst wenn tausende Nutzer wegen Mängeln erfolgreich vom Vertrag zurücktreten würden, so ständen sie danach ohne Lizenz und damit ohne Betriebssystem da. Ihre Investitionen in Microsoft-kompatible Programme wären damit verloren. Fraglich ist, ob diese über § 284 BGB eingeklagt werden könnten. In jedem Fall stellt ein solches Vorgehen ein hohes Risiko dar.

dass der Nutzer die Schadprogramme in der Regel (unwissentlich) selbst installiert. Dem Hersteller verbleibt damit noch die Argumentation, der Nutzer habe durch Nutzung der Software in einer bekanntermassen unsicheren Umgebung (z.B. Internet) erst dazu beigetragen, dass ein Schaden entsteht.

f) Funktion von Open-Source

Ein grosser Teil der Nutzer von Software versucht dieses Problem durch den Einsatz von Open-Source-Software zu lösen. Die Hoffnung dabei ist, dass Schadfunktionen durch die Offenlegung des Quelltextes nicht existieren, zumindest aber schnell entdeckt werden.

g) Zusammenfassung

Es gibt vielfältige Sicherheitsprobleme beim Einsatz von Software. Open-Source wird als Lösungsansatz gesehen.

2. Software als Geschäftsbesorger

Man kann es so betrachten, dass ein Computer mit Software in vielen Bereichen die Besorgung von Geschäften seines Nutzers übernimmt. Nun hat man für Geschäftsführer oder Stellvertreter umfassende rechtliche Regelungen zu Nachfrage-, Sorgfalts- und Rechenschaftspflichten und auch zur Haftung. Diese sind zwar auf die Maschine nicht direkt anwendbar. Doch verwundert es, dass die Hersteller von Software nicht in die Pflicht genommen werden, ihrer Software die Befolgung analoger Grundsätze beizubringen oder anderenfalls selbst zu haften. So führen Webbrowser ohne jede Rückfrage dubiosen Code aus, der Rechner ausspäht oder Festplatteninhalte vernichtet. Würde ein Mensch derartige Tätigkeiten zum Nachteil des Geschäftsherren direkt ausführen, wäre eine Verletzung der Sorgfaltspflichten gegeben. Selbst der mutmassliche Wille des Nutzers ist in diesen Fällen wohl kaum auf die Vernichtung oder Weitergabe der Daten gerichtet. Nur ein Bruchteil der Entscheidungen, die das Programm fällt, lässt sich überhaupt *optional* vom Benutzer beeinflussen. Dem Benutzer müsste es ermöglicht werden, relevante Entscheidungen im Vorhinein oder wenigstens nach Rückfrage durch das Programm vorzunehmen. Die Hersteller von Software statten ihre Systeme indessen mit Fähigkeiten aus, deren Konsequenzen sie selbst nicht überblicken. Das wird sich nicht ändern, sofern die rechtlichen Regelungen nicht angepasst werden.

3. Regelung durch den Markt

Grundsätzlich kann man sich auf den Standpunkt stellen, dass der Markt

allein dies regeln werde. Dazu ist jedoch anzumerken, dass insbesondere der Betriebssystemmarkt derart monopolähnlich strukturiert ist, dass der Verbraucher sich schwerlich mit einer Forderung nach Transparenz durchsetzen kann. Es entstehen ausreichend grosse volkswirtschaftliche Schäden durch Schadprogramme, seien es Viren[1] oder Software, die im Rahmen von Wirtschaftsspionage[2] eingesetzt wird.

4. Gesetzliche Regelung

a) Ziel

Es ist zu prüfen, inwiefern eine gesetzliche Regelung dazu beitragen könnte, das Vertrauen der Nutzer in die von ihnen verwendete Software zu stärken. Software hat mittlerweile in vielen Bereichen eine kritische Bedeutung für die Funktionsfähigkeit der Wirtschaft insgesamt erreicht. Sie kann grundsätzlich zur Effizienzsteigerung und damit höherer Produktivität beitragen. Diese wird gelähmt, wenn die Benutzer auf den Einsatz von Software verzichten oder ggf. auf Open-Source-Lösungen geringeren Leistungsumfangs ausweichen, weil sie unüberwindbare Sicherheitsbedenken haben.

b) Regelungsinhalt

Die Formulierung eines solchen Gesetzes dürfte nicht ganz einfach sein. Der Kern der Regelung könnte sein, dass der Hersteller von Software dafür haftet, dass diese keine Aktionen ohne die (zumindest mutmassliche) Zustimmung des Nutzers ausführt. Das könnte jedoch dazu führen, dass die Software nichts mehr ohne explizite Zustimmung des Benutzers macht und dadurch völlig unproduktiv wird. Eine sinnvolle Restriktion könnte darin bestehen, dass man die Zustimmung auf bestimmte Bereiche eingrenzt. So sind beispielsweise Aktionen des Programmes in seinem eigenen Adressraum weitgehend irrelevant. Sicherheitskritisch sind aber Zugriffe des Programmes auf persistente Daten[3] und die Übertragung von Daten an andere Rechner[4]. Der Zugriff auf andere Software ist primär nicht sicherheitskritisch, da bei dieser wiederum die Zugriffe auf persistente Daten und Übertragung an andere Rechner zustimmungsbedürftig sind. Es könnte jedoch sinnvoll sein, grundsätzlich vom Hersteller jeder Software eine maschinell auswertbare[5]

1. Vgl. z.B. http://www.heise.de/newsticker/data/kav-01.09.01-000.
2. Vgl. http://www.heise.de/ct/99/04/182/ und http://www.sicherheit-im-internet.de/themes/themes.phtml?ttid=57.
3. Z.B. das Schreiben und Lesen von Dateien auf der Festplatte.
4. Z.B. an Rechner im lokalen Netzwerk oder Internet.
5. Diese wäre damit vom Betriebssystem oder spezieller Sicherheitssoftware überprüfbar.

Erklärung darüber zu verlangen, welche Schnittstellen seine Software bedient und zu welchem Zweck[1].

Eine andere Möglichkeit könnte man aus dem Bereich der Buchhaltungssoftware übernehmen: Diese muss faktisch bereits ein GoB-Testat aufweisen, damit das Finanzamt die Buchhaltung anerkennt[2]. Ein ähnliches Testat könnte für die Sicherheit von Software vorgeschrieben werden[3].

c) Parallelen

Eine Parallele hätte eine solche Regelung z.B. in der Deklaration der Inhaltsstoffe bei Lebensmitteln. Auch dort verlangt man aus Verbraucherschutzgründen die Deklaration. Im Bereich der Tabakerzeugnisse ist die Deklaration „Rauchen gefährdet ihre Gesundheit“ eine solche Schutzvorschrift. Auch hier mutet man dem Verbraucher nicht zu, selbst anhand aufwendiger wissenschaftlicher Untersuchungen zu entscheiden, ob ihm ein Schaden droht oder nicht. Im Automobilbereich gibt es die vorgeschriebene TÜV-Untersuchung, die zumindest ein Grundmass an Sicherheit festlegt. Und bei Buchhaltungssoftware hat faktisch jedes verbreitet eingesetzte Programm ein GoB-Testat.

d) Kosten

Den Anbietern entstehen Kosten durch die Zertifizierung. Die spontane Veröffentlichung von Software wird gehemmt. Hobbyentwicklungen werden das Zertifikat aller Voraussicht nicht erhalten. Die Kosten werden auf die Nutzer umgelegt und dürften gerade bei stark verbreiteten Programmen nicht ins Gewicht fallen.

e) Nutzen

Der Verbraucher könnte sich darauf verlassen, dass die Software ein Mindestmass anerkannter Sicherheitsstandards einhält. Ein Testat z.B. durch vom BSI[4] anerkannte Zertifizierungsstellen ermöglicht die Überprüfung auch ohne allgemeine Veröffentlichung des Quelltextes, analog der Tätigkeit von Wirtschaftsprüfern. Ebenso wie die GoB im Bereich der Buchhaltung können diese Regelungen schneller angepasst

1. Also beispielsweise die Schnittstellen des Betriebssystems zum Lesen und Schreiben von Dateien, um auf die Datei „info.txt“ zuzugreifen. Eine derartige Deklaration gibt es bereits bei Java. Dort enthält die Datei java.policy die detailliert aufgeschlüsselten Zugriffsrechte des Programmes. Die Java Virtual Machine überwacht deren Einhaltung, vgl. die Darstellung sog. Permissions in http://java.sun.com/products/jdk/1.2/docs/guide/security/spec/security-spec.doc3.html.
2. Das Testat besagt, dass die Software die Grundsätze ordnungsmässiger Buchführung (GoB) einhält, vgl. http://www.computer-media.de/buchh/b_ges2.htm.
3. vgl. Sandl S. 348.
4. Bundesamt für Sicherheit in der Informationstechnik, vgl. http://www.bsi.de

werden als staatliche Prüfvorschriften. Im Laufe der Zeit erhöht sich die Qualität der am Markt angebotenen Software, da Verbraucher nach dem aussagekräftigen Testat selektieren können. Unternehmen können beispielsweise der Einkaufsabteilung auftragen, nur noch testierte Software zu lizensieren. Die Deklaration der Schnittstellennutzung würde eine automatisierte Überprüfung z.B. durch das Betriebssystem ermöglichen. Die Resourcennutzung erfolgt effizienter, da die einzelnen Verbraucher weniger Aufwand für die Prävention oder Folgenbeseitigung von unauthorisierten Aktionen von Software aufwenden müssen.

f) Zusammenfassung

Es ist möglich und sinnvoll, eine gesetzliche Regelung zum Schutz der Verbraucher vor schadhafter Software zu treffen, da die Vorteile gegenüber möglichen Nachteilen überwiegen. Die Kosten durch Schäden übersteigen die Präventionskosten um ein Vielfaches[1]. Es ist allemal besser, die Kosten der Testierung auf (verhältnismässig wenige) Anbieter zu verteilen, als den Schaden von der Allgemeinheit tragen zu lassen. Zwar reichen die Anbieter die Kosten an die Konsumenten weiter. Das gelingt aber nur den erfolgreichen Anbietern. Die Testatskosten sollten bei den üblichen Kosten von Softwareprojekten auch kaum ins Gewicht fallen. Die Deklaration der Schnittstellennutzung ist technisch unproblematisch und wenig Aufwand zu realisieren[2].

5. Zusammenfassung

Man benötigt nicht unbedingt Open-Source, um Schutz vor unauthorisierten Aktionen von Software zu erreichen. Mit einer Zertifizierung ähnlich den GoB-Testaten kann das Problem effizienter gelöst werden.

IV. Zusammenfassung

Es gibt eine Reihe von Alternativen zum Open-Source-Modell, die eine bessere und systemkompatiblere Lösung der Probleme versprechen und zu einer höheren Effizienz beitragen können.

F. FAZIT

I. Grundannahmen des Urheberrechts

Diese sind nicht zu überdenken[3]. Die unmittelbare Unentgeltlichkeit von Open-Source hindert nicht die Refinanzierung, sondern fördert sie

1. Vgl. zur Schaffung einer öffentlichen Infrastruktur für die digitale Signatur: http://www.heise.de/newsticker/data/em-25.05.02-001.
2. Durch die bereits erwähnte Java-Sandbox ist die Machbarkeit bewiesen.
3. So auch Koch, S. 280.

in den meisten Fällen. Wenn man von eher unbedeutenden altruistischen Beiträgen einzelner Urheber absieht[1], erfolgt eine Refinanzierung durch Querfinanzierung, auch über Unternehmensgrenzen hinweg[2]. Es wurden ferner zahlreiche Anreize zur Entwicklung von Software im Allgemeinen und Open-Source im Besonderen aufgezeigt, die nicht durch das Urheberrecht tangiert werden.

II. Open Source als Modell für Informationsproduktion

Aufgrund der vielfältigen Probleme eignet sich Open-Source nicht als generelles Modell einer Informationsproduktion[3]. Es ist auch bei einer Neugestaltung der Rechtslage nicht davon auszugehen, dass Open-Source herkömmliche Softwarevertriebsmodelle ersetzen könnte[4]. Statt der Lösung über das Open-Source-Modell bieten sich eine Reihe von Alternativen an, um die Effizienz im Softwaremarkt auf Seiten der Urheber und Nutzer zu erhöhen. Open-Source sollte aber als Korrektiv und Experimentierplattform gefördert werden, beispielsweise durch Schutz vor Patentansprüchen[5].

III. Vereinbarkeit mit Urhebervertragsrecht

Open-Source-Lizenzen sind mit dem Urhebervertragsrecht vereinbar[6].

1. Diese erfolgen oft, weil sie für unbedeutend bzw. nicht sinnvoll kommerziell verwertbar gehalten werden.
2. Dienstleistungen, Hardware, Lizenzen für Nicht-Open-Source-Software.
3. So auch Koch, S. 280.
4. Vgl. Jaeger/Metzger, http://www.urheberrecht.org/UrhGE-2000/download/stellungnahmen/urhebervertragsrecht.pdf, S. 1-2, die vom „Parallelmarkt“ sprechen.
5. Vgl. Lessig, S. 222 und 230, der bei Funkfrequenzen einen Mix von fest vergebenen (versteigerten) und frei nutzbaren Frequenzen für wettbewerbsfördernd hält, insbesondere unter dem Aspekt, dass Entrepreneure geringe Markteintrittshürden haben, jedoch später in das „kostenpflichtige“ Segment wechseln können. Man kann dies auch mit der Einteilung in ein „Nichtschwimmerbecken“ und ein „Schwimmerbecken“ vergleichen: Ein Nichtschwimmerbecken sollte es in jedem Schwimmbad geben, um jedem die Möglichkeit zu geben Schwimmen zu lernen und später in das Schwimmerbecken zu wechseln.
6. Vgl. „c) unterliegt dem Urhebervertragsrecht” auf Seite 16.