Sicherheit mit Open Source – Die Debatte im Kontext, die Argumente auf dem Prüfstein

ROBERT A. GEHRING

Einführung

Die Debatte darüber, welches denn nun der sicherere Weg der Softwareentwicklung sei, hält seit einigen Jahren an und wird wohl in absehbarer Zeit auch nicht beigelegt werden – dazu sind die auf dem Spiel stehenden Interessen einfach zu groß. Das Argument der Sicherheit hat in den vergangenen Jahren nicht zuletzt deswegen an Gewicht gewonnen, weil jährlich Milliardenschäden durch Viren, Würmer, Einbrüche in Computersysteme und, nicht zu vergessen, Softwarefehler vermeldet werden. Der Leidensdruck derjenigen, die auf verlässliche IT-Systeme[1] angewiesen sind, wächst stetig, und die Suche nach einem Ausweg gewinnt an Priorität.

Im vorliegenden Beitrag unternimmt der Autor den Versuch, die Debatte um die Sicherheit von Open-Source in ihren Grundzügen zu systematisieren. Ausgehend von der Identifizierung der tragenden Pfeiler der Debatte werden Defizite aufgezeigt und die Potentiale von Open-Source, die Sicherheit von IT-Systemen zu verbessern, insbesondere in den Bereichen Softwaretechnologie und Softwareökonomie diskutiert.

Der 1. Pfeiler: Sicherheit als Marketing-Argument

Open-Source-Software (OSS), so deren Protagonisten, biete einen Ausweg aus der andauernden Softwarekrise. Meinungsäußerungen wie, die Ergebnisse von Open-Source-Softwareentwicklung seien „[g]rundsätzlich besser als andere Software, weil eleganter und obendrein sicherer und zuverlässiger“ (Dignatz 1999 unter Berufung auf Hal Varian) oder „Faster, Better, and Cheaper“ (Scacchi 2002),[2] hört und liest der interessierte Beobachter immer aufs Neue.[3] Andere fragen eher skeptisch „Open Source Security: Opportunity or Oxymoron?“ (Lawton 2002)

Man darf in diesem Zusammenhang jedenfalls nicht übersehen, dass aus dem, was 1984 als Richard Stallmans gegen die mit zunehmender Kommerzialisierung eingehenden „enclosures“[4] bei Software gerichtete Ein-Mann-Aktion des „last true

---

1 „IT-Systeme“ wird hier in einem generischen Sinne verwandt und beschreibt funktionale Konglomerate aus Hard- und Software, die in der Regel Netzwerkelemente zum Datenaustausch mit anderen IT-Systemen beinhalten.

2 In der Fachzeitschrift Wirtschaftsinformatik gab es kürzlich einen Schwerpunkt zum Thema „Sicherheit durch Open Source“ (Jg. 45, 2003, Nr. 4, S. 474–482, online: http://www.wirtschaftsinformatik.de/wi_heft.php?op=heft&heftid=140). Anhand der dort vorgestellten Beiträge kann man sich einen guten Überblick über das Meinungsspektrum verschaffen.

3 Nicht immer stehen hinter solchen Meinungen um Objektivität bemühte Wissenschaftler. Man wird auch eine ganze Reihe tendenziell gegen Microsoft und (oder) sein Software-Monopol eingestellter Interessensgruppen zum Chor der „Open-Source ist sicherer“-Protagonisten zählen dürfen.

4 Der Terminus „enclosure“ beschreibt ursprünglich die Einhegung vormals frei zugänglichen Weide-

hacker left on earth" (Levy 2001, S. 427) begann, mittlerweile ein „Big Business"[5] geworden ist, in dem Milliardenumsätze getätigt werden.[6] Da „gehört Klappern zum Handwerk", will man sich gegen die Konkurrenten aus dem Lager der Closed-Source-Software (CSS) durchsetzen; schließlich wird ja auch ein Open-Source-Antagonist wie Microsoft nicht müde, seine Direktoren und Verkäufer das Argument der Sicherheit bzw. Vertrauenswürdigkeit (*trustworthiness*, wie der neuere Microsoft-Terminus lautet) im Munde führen zu lassen.[7]

Der 2. Pfeiler: Sicherheit und Politik

Ein weiterer Faktor, der die Debatte um die Sicherheit von Open-Source befeuert, ist geostrategischer Natur. In der von US-Unternehmen dominierten PC-Softwarewelt der letzten 30 Jahre war es Unternehmen in anderen Ländern nur ausnahmsweise gelungen, sich zum „global player" aufzuschwingen. SAP ist sicherlich das bekannteste Beispiel. In den meisten Fällen dominieren die von US-Unternehmen stammenden Softwareprodukte derartig den internationalen und lokalen Markt, dass ein echter Wettbewerb nicht in Gang kommen konnte.[8] Die rigorose Durchsetzung von Intellectual Property Rights (IPRs) amerikanischer Copyrightindustrien (inkl. Softwarehersteller) auf internationaler Ebene,[9] hat maßgeblich dazu beigetragen, die Positionen der US-Software-Industrie zu stärken. US-amerikanische Software dominiert in Indien, Japan, Korea, China, usw. die lokalen Märkte, und sei es in Form von illegalen Kopien. Nicht alle von diesen Ländern standen in einem freundschaftlichen Verhältnis zu den USA, sei es aus politischen (China) oder vorrangig aus wirtschaftlichen Gründen (etwa Indien und Japan). Man argwöhnte dort,

landes („commons"), wie sie etwa seit dem 15. Jahrhundert in England stattfanden und unter dem Namen „fencing" aus den USA des 19. Jahrhunderts bekannt sind. In den letzten Jahren wurde „enclosures" von verschiedenen Autoren auch zunehmend auf Entwicklungen im Cyberspace angewandt.

5 So fragte ein Bericht der Deutsche Bank Research (Hofmann 2002). „One commentator has argued that ‚anarcho-communism is now sponsored by corporate capital.'" (Bollier 2003, S. 37, Fußnote ausgelassen). Dieser Umstand findet, wie Richard Stallman nicht müde wird, zu betonen, nicht seine ungeteilte Zustimmung. Siehe auch die Aussagen des GNU-Projekts in „Why ‚Free Software' is better than ‚Open Source'", http://www.gnu.org/philosophy/free-software-for-freedom.html (08. Jan 2004).

6 Laut Financial Times Deutschland vom 29.12.2003 wurde mit Linux-basierten Servern im Jahr 2002 ein Umsatz von 1,4 Mrd. US$ erwirtschaftet. Allein in den ersten drei Quartalen des Jahres 2003 stieg der Umsatz auf 2,2 Mrd. US$. IBM allein meldete für 2002 einen Umsatz im Linux-Geschäft von 1,5 Mrd. US$ und HP nannte 2 Mrd. US$. (Goltzsch 2003)

7 Sicherheit stünde für die nächsten „drei bis vier Jahre auf unserer Prioritätenliste ganz oben," äußerte sich Bill Gates (heise 2004). Im Streit um den Zusammenhang zwischen Offenheit des Quellcodes und die Sicherheit von Software beschwichtigt Microsoft: „Development models ultimately have a neutral effect on software security." (Mundie 2003)

8 Diese Dominanz ist, wie im Fall von Microsoft gerichtlich festgestellt wurde, zu einem gewichtigen Teil durch wettbewerbswidriges Verhalten zustandegekommen. Vgl. Ishii und Lutterbeck (2002).

9 Auf Drängen der Vertreter aus den „Copyright-Industrien", Softwarehersteller eingeschlossen, wurde 1984 Abschnitt 301 des US-Trade Acts von 1974 dahingehend ergänzt, dass die mangelhafte Durchsetzung von IPRs in einem Land als „unfair trade barrier that could provoke U.S. retaliation" (Ryan 1998, S. 11) betrachtet wurde. Bei den Verhandlungen der Uruguay-Runde zum Welthandelsabkommen GATT überraschte der US-Handelsbeauftrage 1986 mit dem Ansinnen, dass IPRs unbedingt in die Verhandlungen einzubeziehen seien (Ryan 1998, S. 1). Ein brasilianischer Beobachter schildert das Ergebnis so: „Industrialized countries forced developing countries to initiate

dass US-Software auf einheimischen Computern nicht unbedingt vertrauenswürdig sei. Die Crypto-Debatte der 90er Jahre[10] um starke und schwache Verschlüsselung, Exportverbote,[11] Schlüsselhinterlegung und Hintertüren in Software, der Versuch der US-Regierung, mit dem Clipper-Chip eine definitiv abhörbare Technologie in den Markt zu drücken usw., stießen gerade auch bei Sicherheitsfachleuten auf scharfe Kritik, die zur damaligen Zeit in Kryptographie noch die Lösung für alle Sicherheitsprobleme im IT-Bereich sahen.[12] Auch die Versuche der US-Administration, in internationalen Verhandlungen Einschränkungen für den Einsatz von Verschlüsselungsverfahren durchzusetzen, trafen auf Missfallen, nicht zuletzt in Europa, wo man durchaus ernste Befürchtungen hinsichtlich High-Tech-Spionage hegte. Das Bekanntwerden des von den USA und einigen engen Verbündeten (Großbritannien, Australien, Neuseeland) betriebenen Abhörnetzwerkes Echelon[13] trug ein Übriges dazu bei, ein allgemeines Misstrauen gegenüber den USA zu stützen, das sich auch auf die von dort stammende Software erstreckte. Da man in den streng geheimgehaltenen Code ja nicht hineinsehen konnte, war man auch nicht in der Lage, dessen Vertrauenswürdigkeit zu beurteilen. Das war ein klares Argument *contra* Closed-Source-Software, das in der Folge zu einem Argument *pro* Open-Source-Software wurde: Bei offen gelegtem Quellcode kann sich im Prinzip jede/r davon überzeugen, ob die Software etwa auf der eigenen Festplatte spioniert und „nach Hause telefoniert." Dieses Argument spielte in der deutschen Debatte immer wieder eine große Rolle[14] und verfehlte auch anderswo seine Wirkung nicht. So haben diverse asiatische Regierungen ein verstärktes Engagement im Open-Source-Bereich entwickelt.[15]

---

negotiation of an agreement on TRIPS with the clear objective of universalizing the standards of IPRs protection that the former had incorporated in their legislation [...]" (Correa 2000). Die Vorgänge, die zur Verabschiedung von WIPO Copyright Treaty (WCT) und WIPO Performances and Phonograms Treaty (WPPT) im Dezember 1996 führten, zeigen eine ähnliche Entwicklungsgeschichte (Ryan 1998; Litman 2001; Drahos und Braithwaite 2002). Auch bei der Verabschiedung der gültigen Fassung der EU-Softwarerichtlinie (91/250/EWG) spielte die globale Auseinandersetzung in den „Softwars" (Clapes 1993) zwischen den USA und ihren von ihnen wahrgenommenen Wettbewerbern (EU, Japan) eine wesentliche Rolle.

10 Die Crypto-Debatte und ihre Vorgeschichte wird u.a. dargestellt in Diffie und Landau (1998).

11 Siehe etwa auch die Prozesse Bernstein v. United States Department of Justice, 176 F.3d 1132 (9th Cir. 1999) (Lemley, Menell, Merges und Samuelson 2002, S. 914–915) und Junger v. Daley, 209 F.3d 481 (6th Cir. 2000) (Maggs, Soma und Sprowl 2001, S. 819–823).

12 Der Irrtum, mit Kryptographie die Sicherheitsprobleme in vernetzten IT-Systemen lösen zu können, ist mittlerweile von vielen Fachleuten, nicht zuletzt von Bruce Schneier, erkannt worden: „In my vision cryptography was the great technological equalizer; anyone with a cheap (and getting cheaper every year) cornputer could have the same security as the largest government. In the second edition of the same book, written two years later, I went so far as to write: ‚It is insufficient to protect ourselves with laws; we need to protect ourselves with mathematics.' It's just not true. Cryptography can't do any of that. [...] Security, palpable security that you or I might find useful in our lives, involves people: things people know, relationships between people, people and how they relate to machines. Digital security involves computers: complex, unstable, buggy computers." (Schneier 2000, S. xi)

13 Vgl. Schulzki-Haddouti (2001).

14 Vgl. z. B. Köhntopp, Köhntopp und Pfitzmann (2000).

15 Einem Artikel des Economist vom September 2003 zufolge „China has been working on a local version of Linux for years, on the grounds of national self-sufficiency, security and to avoid being

Mit der zunehmenden Verfügbarkeit von Open-Source-Software in der ganzen Welt und ihrer wachsenden Leistungsfähigkeit, wird nicht nur die Abhängigkeit aller Länder von US-geliefertem Code reduziert, sondern es verliert das IPR-Paradigma hinter dem Closed-Source-Software-Modell selbst spürbar an Überzeugungskraft. Wie David Bollier es ausdrückt:

> [O]nline sharing and collaboration raise some profound questions about the foundations of intellectual property law, which assumes that useful creativity will emerge only through a sanctioned structure of markets, incentives, and contracts. (2003, S. 208)

Mit dem Einsatz von Open-Source-Software lösen die jeweiligen Länder nicht nur Ihr Sicherheitsdilemma (Vertrauensdilemma), sondern versetzen sich selbst in die Lage, eine eigene, konkurrenzfähige Softwareindustrie zu entwickeln. Die Reichweite der Verquickung des offensichtlichen Arguments (Sicherheit und Vertrauenswürdikeit) mit dem ökonomischen Argument (industrielle Eigenständigkeit) sollte in seiner Bedeutung nicht unterschätzt werden. Damit ordnet sich der Streit um die Sicherheit von Softwareentwicklungsmodellen ein in die weitaus größere Auseinandersetzung um die Frage, wie denn die künftige Verfasstheit der Informationsgesellschaft aussehen solle, und welche Rolle exklusive Eigentumsrechte dabei spielen werden[16] – nicht nur, aber insbesondere auch für Software.[17]

Der 3. Pfeiler: Sicherheit und Softwaretechnologie

Aus rein softwaretechnologischer Sicht wurden ebenfalls Argumente zugunsten eines Open-Source-Ansatzes vorgebracht. Das Open-Source-Entwicklungsmodell sei besser, so verlautet immer wieder, weil die so entstehende Software weniger Fehler aufweisen würde. „Given enough eyeballs, all bugs are shallow," formulierte Eric S. Raymond (1999, S. 41) in seinem berühmten Aufsatz „The Cathedral and the Bazaar" die These, dass die Offenlegung des Codes den Beta-Testern eine gründliche Inspektion im Sinne des wissenschaftlichen *peer reviews* ermöglichen würde.[18] Fehler

too dependent on a single foreign supplier." In Korea ist die Regierung bemüht, Microsoft-Software durch Open-Source-Produkte zu ersetzen (Myung 2003). Und in Indien hat gar der Präsident persönlich dazu aufgerufen, Open-Source-Software zu entwickeln (Kalam 2003). Dass gerade Asien so offen ist für Open-Source, könnte vielleicht z.T. damit erklärt werden, dass dem Lernen durch Imitieren und Verbessern im Konfuzianismus (der in Asien über die die Jahrhunderte bekanntlich eine enorme Wirkung entfaltete) eine Schlüsselstellung zukommt (Wei-Ming 1993).

16 Vgl. u.a. National Research Council (1991), Office of Technology Assessment (1992), Branscomb (1994), Lessig (1999), Imparato (1999), National Research Council (2000) und Bollier (2003) sowie Thierer und Wayne, Jr. (2003) (stellvertretend für die konservativen Auffassungen in den USA) und McChesney, Wood und Foster (1998) (stellvertretend für die Auffassungen der sozialistisch orientierten Kreise in den USA).

17 Siehe auch Lutterbeck, Horns und Gehring (2000).

18 Nach Mustonen (2003, S. 103) hat eine Reihe von Autoren argumentiert, dass neues Wissen in der arbeitsteiligen Gesellschaft typischerweise unter zwei unterschiedlichen Anreizstrukturen generiert wird. Neben der profitgesteuerten „technology" gibt es die Anreizstrukturen innerhalb von „science". Dort spielen „peer recognition" und die aus einer schnellen Veröffentlichung von neuem Wissen zu erzielende Reputation die Rolle der maßgeblichen Stimulationsfaktoren: „Scientific recognition is achieved by making one's contribution public to peer review as quickly as possible and acquiring priority to the new knowledge." (Mustonen 2003, S. 103) Der Aspekt des *peer reviews* in der Open-Source-Entwicklung wurde genauer untersucht von Stark (2002), die eine positive Attitüde

würden daher sehr schnell entdeckt und beseitigt werden. Da viele Fehler aber gleichzeitig Sicherheitslücken darstellen würden, wären Open-Source-Programme sicherer, da sie weniger Fehler hätten. Eine solche Schlussfolgerung hat Raymond in seinem Aufsatz zwar selbst nicht unmittelbar gezogen, sie lag aber nahe,[19] ist der Zusammenhang zwischen bestimmten Fehlertypen und Sicherheitsmängeln durch langjährige Erfahrungen doch empirisch gut belegt. Immer wieder führen etwa so genannte Pufferüberläufe (buffer overflows), d.h. klassische Programmierfehler, zu verletzlichen Computersystemen, auf die erfolgreiche Angriffe möglich sind.[20] Zwar gibt es keine 1:1-Beziehung zwischen Softwarefehlern und Sicherheitslücken, aber beide Größen sind eng miteinander verknüpft,[21] was sich auch darin zeigt, dass in der Diskussion Begriffe wie „dependable", „robust", „trustworthy" und „reliable" alle mit einer Konnotation von Sicherheit daherkommen.[22] Andere Argumente im Sinne der Sicherheit durch Open-Source-Entwicklung sind die Möglichkeit, die Komplexität der Software zielgerichtet durch Entfernung von Code reduzieren zu können, sowie die durch die verteilte Entwicklung bedingte Modularität[23] des Codes. Die Stellungnahmen der meisten Fachleute zu diesen Argumenten bleiben allerdings insgesamt recht wage:

> Open source software can be potentially more secure than closed source software, in which source codes of programs can be examined by Peer Review from parties other than the creators. However the openness of source codes is not necessary an advantage by itself, the level of software security further depends on the efficiency of Peer Review,

hinsichtlich des *peer reviews* (bei Open-Source-Entwicklern) nachgewiesen hat.

19 Sie wurde von verschiedenen Autoren im Laufe der letzten Jahre immer wieder gezogen, zuletzt von Oppliger (2003).

20 Zum Problem der Verletzlichkeit durch „buffer overflows" siehe z.B. Krsul, Spafford und Tripunitara (1998). Zur Bedeutung in der Praxis siehe Kamerling (2003), wo die 20 kritischsten Schwachpunkte für Windows- und UNIX-Betriebssysteme aufgelistet werden. Unter den kritischsten Schwachpunkten findet sich eine ganze Reihe, die auf „buffer overflows" zurückzuführen ist.

21 Eine relativierende Formulierung ist notwendig, da es keinen wissenschaftlich formulierten Zusammenhang zwischen Fehlern in und Sicherheit von Software, sondern lediglich Erfahrungswerte gibt. Sichere Software muss nicht fehlerfrei sein, fehlerfreie Software kann unsicher sein, und in den meisten Fällen wird man davon ausgehen müssen, dass die Software weder fehlerfrei noch sicher ist, wobei es in der Regel noch nicht einmal Einigkeit hinsichtlich der Terminologie gibt (Jonsson, Strömberg und Lindskog 2000). Das viel grundlegendere Problem aber ist, dass es bis dato keine wissenschaftlich fundierten Verfahren zur Entwicklung von sicherer Software gibt (Greenwals u.a. 2003), von fehlerfreier Software ganz zu schweigen. Der alternative Ansatz, Software-Code nicht zu schreiben, sondern nach formalen Methoden zu entwickeln, d.h. aus einer formal vollständigen Spezifikation zu generieren, hat seinerseits Tücken. Formale Methoden sind z.B. oft unökomisch (Luqi und Goguen 1997).

22 Vgl. u.a. Neumann (1998, S. 128): „‚[R]obust' is an intentionally inclusive term embracing meaningful security, reliability, availability, and system survivability, in the face of a wide and realistic range of potential adversities[...]"; Littlewood und Strigini (2000, S. 177): „Dependability encompasses, among other attributes, reliability, safety, security, and availability."; Knight (2002, S. 685) „The notion of dependability is broken down into six fundamental properties: (1) reliability; (2) availability; (3) safety; (4) confidentiality; (5) integrity; and (6) maintainability. [...] It is important to note that security [...] is not included explicitly in this list. Rather, it is covered by the fourth and fifth items in the list.". Siehe weiterhin Anderson (2001), Viega und McGraw (2002), Meadows und McLean

the skill of reviewers and consistent support of software after the initial release." (Li 2002, S. 73)

Der 4. Pfeiler: Sicherheit und Software-Ökonomie

Software existiert nicht in einer idealen Welt, sondern wird von realen Menschen entwickelt und in einer realen Welt verteilt und eingesetzt. Die reale Welt ist eine Marktwirtschaft in der Software oft mit knappen Ressourcen produziert wird und als Ware den Bedingungen des Marktes genügen muss.[24] Dass das Sicherheitsargument regelmäßig dazu dient, Open-Source-Software zu vermarkten, mag als (schwacher) Beleg dafür gelten.

Der Zusammenhang zwischen Softwarequalität und Softwareökonomie im Allgemeinen wird, im Gegensatz zum technologischen Zusammenhang, erst wieder in jüngerer Zeit diskutiert und untersucht.[25] Ausgangspunkt ist dabei üblicherweise eine mikroökonomisch fundierte Kosten-Nutzen-Betrachtung: Es erfordert nicht unerhebliche Investitionen, Systeme sicherer zu machen, d.h. das mit ihrem Einsatz verbundene Risiko zu senken. Wenn die notwendigen Investitionen höher ausfallen als das in einem möglichen Schadensfall realisierte Risiko, ist es ökonomisch sinnvoller, die Investitionen nicht zu tätigen: „An economics perspective naturally recognizes that while some investment in information security is good, more security is not always worth the cost." (Gordon und Loeb 2002) Stattdessen könnte etwa ein Teil davon in die Absicherung von anderen Risiken mit höherem Schadenspotential fließen.[26]

Weitgehend unberücksichtigt blieben bisher Externalitäten, d.h. das Problem, dass das Schadensrisiko nicht notwendig bei demjenigen eintritt, der sein System nicht absichert. Man denke etwa an Internetserver die „gehackt" und als Viren- bzw. Spamverteiler genutzt werden.[27] In diesem Falle entsteht der überwiegende

(1999) und Parnas, van Schouwen und Kwan (1990).

23 Den Zusammenhang zwischen Modularität, Code-Umfang und Softwarequalität diskutieren z.B. Stamelos, Angelis, Oikonomou und Bleris (2002).

24 Zu der Frage, inwiefern auch Open-Source-Software i.A. als Ware betrachtet werden muss, siehe den Beitrag von *Heller* und *Nuss* im vorliegenden Buch.

25 Vgl. u.a. Anderson (2001a), Gordon und Loeb (2002), Hoglund (2002), Sandhu (2003) sowie schwerpunktmäßig WEIS (2002; 2003). Bereits in Boehm (1981) wurde der Zusammenhang diskutiert, geriet zwischendurch aber – trotz gelegentlicher Publikationen zum Thema (z.B. Levy 1987) – weitgehend in Vergessenheit, wie auch beklagt wurde: Die Ökonomie sei „A Missing Link in Software Engineering," so der Titel eines Beitrages von Tockey (1997).

26 Die Frage der richtigen Investitionen in IT-Sicherheit wurde bis dato noch nicht zufriedenstellend beantwortet und es gibt diesbezüglich einige bemerkenswerte Widersprüche. Am 13. Februar 2002, zum Beispiel, legte das Office of Management and Budget dem US-Kongress einen Bericht vor, in dem die Ausgaben der US-Regierung für IT-Sicherheit evaluiert wurden. Der dortige Befund ist bemerkenswert: „Among other findings, the report stated they found no evidence of correlation between the amount spent on infomration security and the level of security performance." (Carini und Morel 2002).

27 Zum Zeitpunkt der Fertigstellung dieses Aufsatzes kursieren im Internet gerade zwei Varianten eines Wurms namens „MyDoom", deren Ausbreitungsgeschwindigkeit und -reichtweite alles bisher Dagewesene in den Schatten stellt. Der Wurm kommt als Anhang einer E-Mail mit gefälschtem Absender und irreführender Betreffzeile und erweckt den Anschein einer fehlgeleiteten E-Mail. Nach Öffnen des Anhangs auf einem Microsoft-Betriebssystem installiert der Wurm eine Hintertür im PC und bereitet diesen darauf vor, in einem konzertierten Angriff Anfang Februar die Webserver von

Schaden gar nicht beim Serverbetreiber, sodass dessen Anreize, in die Serversicherheit zu investieren begrenzt sind – zumal er für den so angerichteten Schaden in der Regel auch nicht haftbar gemacht werden kann. Zwar wurde zur Lösung dieses Problems gelegentlich Versicherungsmodelle[28] oder (neue) Haftungsregelungen[29] ins Spiel gebracht, ohne dabei allerdings die Besonderheiten des Haftungsrechts hinsichtlich des Schadensnachweises und der dafür notwendigen Beweisführung in Sachen Ursächlichkeit zu berücksichtigen.

Ein Vorschlag der seit Jahren immer wieder einmal auftaucht ist, Softwareentwickler zu lizenzieren,[30] um so zu erreichen, dass nur „gute" Softwareentwickler Software schreiben, deren durchschnittliche Qualität dann natürlich steigen würde. Allerdings hält sich die Begeisterung für diesen Vorschlag in Grenzen und die Resultate des Open-Source-Modells scheinen seiner Sinnfälligkeit auch zu widersprechen.

Nur wenige Autoren haben sich bereits mit Fragen des Zusammenhangs zwischen den Spezifika der Software-Ökonomie (als konkreter Anwendung der Informationsökonomie) und ihren Auswirkungen auf die Sicherheit von OSS befasst. Zu nennen sind in dieser Hinsicht Landwehr (2002), Gehring (2003) sowie Franke und von Hippel (2003).[31] Der Tenor aller genannter Beiträge lautet dahingehend, dass die Open-Source-Methode mit ihren charakteristischen Eigenschaften,

- den Informationsfluss zwischen Programmierern und Anwendern zu verbessern, und
- den Anwendern die (rechtlichen und technischen) Mittel an die Hand zu geben, das System in seiner Substanz, im Quellcode, zu ändern,

einen positiven Beitrag zur Sicherheit leistet. Diesen Auffassungen zu Grunde liegt die Feststellung, dass dem proprietären Modell der Softwareentwicklung und -vermarktung Schwächen inhärent sind,[32] die hinsichtlich der Produktqualität unter dem Blickwinkel von Sicherheit wohl nicht innerhalb dieses Modells überwunden werden können. Open-Source bildet dann einen Ausweg aus der Perspektive des Anwenders.

Kritik: Defizite der Debatte

Weitgehend übersehen[33] wurde bisher ein weiteres Argument, jedenfalls insofern es um den Beitrag zu Softwarequalität und -sicherheit geht: der Faktor Mensch

SCO und Microsoft lahmzulegen. Die infizierten Rechner selbst werden vorerst nicht in Mitleidenschaft gezogen, allerdings verheisst die Hintertür für die Zukunft nichts Gutes.

28 Für eine Versicherungslösung als „A Market Solution to the Internet Security Market Failure" plädieren z. B. Yurcik und Doss (2002).

29 Für eine Haftungslösung sprechen sich z.B. Fisk (2002) und Schneier (2002) aus; Martin (2002) äußert sich skeptisch hinsichtlich der Haftungsvorschläge.

30 Dafür z. B. Dorchester (1999), Frailey (1999), Tripp (2002); dagegen z.B. Kolawa (2002).

31 Anderson (2003) spricht das Thema zwar auch – aus theroretischer Perspektive – an, geht aber nicht weiter auf die Praxis ein und fordert statt dessen mehr empirische Untersuchungen. Ähnliches gilt für Lutterbeck, Horns und Gehring (2000).

32 Das betrifft etwa das Missverhalten, neue Produkte auf den Markt zu bringen (um neue Einnahmen zu generieren), ohne die Fehler in bereits vermarkteten Produkten vorher zu beseitigen.

33 Die Ausnahme bildet eine kurzen Erwähnung Weinbergs und des „egoless programming" (s.u.) bei Raymond (1999, S. 61).

oder, mit anderen Worten, Psychologie und Soziologie[34] der arbeitsteiligen Softwareentwicklung. So spielt das Klima in der Projektorganisation eine wesentliche Rolle bei der Aufdeckung und Weitermeldung von Problemen während der Durchführung von Softwareprojekten (Tan, Smith, Keil und Montalegre 2003).

Bereits zu Beginn der 1970er Jahre sind die kognitiven Grenzen des individuellen Softwareentwicklers und die daraus zu ziehenden Schlussfolgerungen für die Arbeitsteilung bei der Softwareentwicklung betont worden. In seinem zeitlosen Standardwerk von 1971 beschreibt Gerald M. Weinberg „Programming as a Social Activity."[35] Komplexe Programme werden, Ausnahmen mögen die Regel bestätigen, in einem Prozess sozialer Interaktion entwickelt, betont Weinberg (1971, S. 52).[36] Als grundsätzlich schädlich betrachtet er hingegen – aus softwaretechnischer Perspektive – das Besitzstandsdenken bei Software:

> Well, what is wrong with ‚owning' programs? Artists ‚own' paintings; authors ‚own' books; architects ‚own' buildings. Don't these attributions lead to admiration and emulation of good workers by lesser ones? (S. 53)

Weinberg verneint die suggestiv naheliegende Schlussfolgerung und weist darauf hin, dass „Fehler" in klassischen Kunstwerken allenfalls eine Geschmackssache seien. Bei Software hingegen entscheidet letzten Endes der Computer, was ein Fehler ist – durch Fehlfunktionen des Programmes. Wo ein Künstler das Urteil seiner subjektiven Kritiker im Prinzip nach Belieben verwerfen kann, ist der Programmierer mit einer „objektiven Kritik" konfrontiert, aus deren Urteil er, wenn ein Fehler auftrit, im Grunde nur eine Konsequenz ziehen kann:

> This program is defective. This program is part of me, an extension of myself, even carrying my name. I am defective. (S. 54)

---

34 Zwar gibt es mittlerweile eine ganze Reihe soziologischer Untersuchungen zu Open-Source-Prozessen, vgl. etwa Robles, Scheider, Tretkowski und Weber (2001), FLOSS (2002) und Hertel, Niedner und Herrmann (2003), doch konzentrieren diese sich vorrangig auf Fragen von Herkunft, Motivation, Qualifikation usw. Die dabei gewonnenen Erkenntnisse sind hinsichtlich der Einflüsse der Soziologie auf Softwarequalität und -sicherheit wenig aussagekräftig, bzw. wurden dahingehend nicht weiter ausgewertet. Zum Einfluss des Betriebsklimas auf Softwareprojekte siehe z.B. Tan, Smith, Keil und Montalegre (2003).

35 So der Titel von Teil 2 in Weinberg (1971).

36 Diese unter dem Paradigma des Software-Engineering ein wenig in Vergessenheit geratene Einsicht ist in jüngerer Zeit zu neuen Ehren gekommen, z.B. in Hohmann (1996). Unter dem Schlagwort vom „agile programming" wird die Bedeutung der sozialen Interaktion für eine erfolgreiche Projektdurchführung mit Emphase betont. Vgl. z.B. Beck (2000) und Cockburn (2002).

Diese Erfahrung stellt den Programmierer vor ein Dilemma kognitiver Dissonanz[37]: Fehler in Programmen verletzen das Selbstwertgefühl des Entwicklers – mit Folgen für die Qualitätssicherung:

> A programmer who truly sees his program as an extension of his own ego is not going to be trying to find all the errors in that program. On the contrary, he is going to be trying to prove that the program is correct – even if this means the oversight of errors which are monstrous to another eye. All programmers are familiar with the symptoms of this dissonance resolution – in others, of course. (S. 55)

Diese Erkenntnisse wurden von Praktikern, wie etwa dem IBM-Forscher Glenford J. Myers (1976; 1979), bestätigt und in Konzeptionen für das erfolgreiche Entwickeln und Testen von Software (mit dem Ziel, die „software reliability" zu steigern) berücksichtigt. Testverfahren wurden so gestaltet, dass sie dem von Weinberg propagierten Ideal des „egoless programming"[38] (1971, S. 56) nahe kamen, das Myers (1976, S. 143) so beschreibt:

> Rather than being secretive and defensive toward his program, the programmer adopts the opposite attitude: his code is really a product of the entire project and he openly asks other programmers to read his code and constructively criticize it. When someone finds an error in his code, the programmer of course should not feel good about making a mistake, but his attitude is, "Wow, we found an error in our product! Good thing we found it now and not later! Let's learn from this mistake and also see if we can find another." A programmer who finds an error in another programmer's code does not respond with, "Look at your stupid mistake"; instead the response is more like "How interesting! I wonder if I made the same mistake in the module I wrote." Notice the fine point concerning cognitive dissonance in the previous sentence: I used the phrase "the module I wrote" instead of "my module".

Wohl nicht zufällig sind die Gemeinsamkeiten mit dem wissenschaftlichen *peer review*.[39] Man erkennt in der Beschreibung von Myers ohne weiteres dieselbe Attitüde wieder, Code nicht als Eigentum, sondern als etwas Gemeinsames, als Extension eines „commons" zu begreifen, wie sie aus den Selbstbeschreibungen der Open-Source-Community hinlänglich bekannt ist. Folgt man Weinberg (1971), Myers

---

37 Unter kognitiver Dissonanz versteht man die „Unvereinbarkeit von mehreren Überzeugungen, Einstellungen, Haltungen gegenüber Umweltsituationen, anderen Menschen u. deren Anschauungen, den eigenen Verhaltensnormen oder Wertmaßstäben u.a." (Hillmann 1994, S. 419). Eine Situation, die kongitive Dissonanz auslöst, wird als bestrafend empfunden, was Versuche des Ausweichens und Verdrängens nach sich zieht. (a.a.O.) Wenn also, begünstigt durch die propagierten Wertauffassungen und das soziale Umfeld, Fehler in Programmen als Bestrafung des eigenen Verhaltens (des Programmierers) angesehen werden oder nach sich ziehen, wird der Programmierer das Bekanntwerden von Fehlern vermeiden wollen, sogar sich selbst gegenüber (indem sie „übersehen" werden): „Programmers, if left to their own devices, will ignore the most glaring errors in their output – errors that anyone else can see in an instant." (Weinberg 1971, S. 56)

38 Weinberg (1971, S. 56) führt einen der Urväter des modernen PCs, John von Neumann, als Kronzeugen für die Vorteile des „egoless programming" an: „John von Neumann himself was perhaps the first programmer to recognize his inadequacies with respect to examination of his own work. Those who knew him have said that he was constantly asserting what a lousy programmer he was, and that he incessantly pushed his programs on other people to read for errors and clumsiness."

39 Zur Bedeutung des *peer review* in der Wissenschaft vgl. z.B. Macrina (1995, S. 69–95).

(1976; 1979) und anderen, so lassen sich Einsichten in den Zusammenhang zwischen Softwareentwicklungsprozess und Produktqualität gewinnen, die eine Fokussierung auf Software als „geistiges Eigentum" (gemäß dem herrschenden Autorenparadigma[40]) aus softwaretechnischer Perspektive grundsätzlich in Frage stellen könnten.[41] Software ist eben kein Buch, das man sich ins Regal stellt, auch wenn es im Wortlaut des deutschen Urheberrechts bloß als ein Sprachwerk unter anderen, wie „Schriftwerke" und „Reden", eingestuft wird (UrhG §2 Abs. 1 Zif. 1).

Diese Ausführungen zu Soziologie und Psychologie der Softwareentwicklung sollen soweit genügen, um eines zu zeigen: In der Debatte um Open Source vs. Closed Source gibt es noch eine ganze Reihe nicht ausgeleuchteter Gassen und Winkel die näherer Untersuchungen wert sind.

Summa: Die Debatte um Open-Source und Sicherheit

Fassen wir einmal kurz zusammen. Vier Pfeiler tragen bisher die Debatte um die Sicherheit von Open-Source-Software:

1. Sicherheit als Verkaufsargument;
2. Zugang zum Quellcode als Voraussetzung für Sicherheit und Vertrauenswürdigkeit – das politische Argument;[42]
3. Open-Source-Entwicklungsmethodik als weniger fehlerträchtig – das technologische Argument;
4. Open-Source als alternativer Ansatz zur Softwareproduktion und -distribution – das ökonomische Argument.

Wer hat nun Recht, wenn es um die Sicherheit von Open-Source geht? Welche Argumente sind stichhaltig(er)?

---

40 Zur Problematik des Autorenparadigmas an und für sich vgl. Boyle (1996).

41 In den Anfangszeiten, als im Hinblick auf Software von geistigem Eigentum noch nicht die Rede war, gelangten einige der ersten Softwareentwickler überhaupt, die Wissenschaftler des EDSAC-Projekts an der Universität Cambridge in England, nach den ersten Erfahrungen mit Bugs (d.h. Fehlern in Programmen) und Debugging (d.h. dem Prozess der Fehlerbeseitigung) zu einer mit dem Autorenparadigma (der IPR-Regime) ziemlich unvereinbaren Auffassung:
„The Cambridge Group decided that the best way of reducing errors in programs would be to develop a ‚subroutine library' – an idea that had already been proposed by von Neumann and Goldstine. It was realized that many operations were common to different programs – for example, calculating a square root or a trigonometrical function, or printing out a number in a particular format. The idea of a subroutine library was to write these operations as kind of miniprograms that programmers could then copy into their own programs. In this way a typical program would consist of perhaps two-thirds subroutines and one-third new code." (Campbell-Kelly und Aspray 1996, S. 185 f.)
In diesem Sinne sollte Programmieren also überwiegend („two-thirds") aus Kopieren bestehen, um die Qualität zu sichern. Jedes Programm wäre somit weniger Ausdruck der Individualität seines Entwicklers als vielmehr Resultat des Gebrauchs sozial hergestellter Erkenntnisse und Artefakte.

42 Im Zusammenhang mit elektronischen Wahlsystemen hat dieses Argument in jüngster Zeit, insbesondere nach Bekanntwerden eklatanter Schwächen in proprietären Closed-Source-Systemen, enorm an Bedeutung gewonnen. Vgl. z.B. Zetter (2004). Manche Fachleute sind allerdings der Meinung, dass softwarebasierte Wahlsysteme grundsätzlich als unsicher zu erachten seien. Entsprechend äußert sich z.B. David Dill: „As a computer scientist, I don't think they can make it secure enough, no matter what their procedure, or how they design the machine, or how the machines are inspected at independent laboratories." (Jonietz 2004, S. 74).

Aus praktischen Gründen, ist es an dieser Stelle nicht möglich, die gesamte verfügbare Literatur auszuwerten, wirklich alle Argumente abzuwägen. Stattdessen wird die weitere Untersuchung im Anschluss auf zwei der genannten Pfeiler fokussiert: Entwicklungsmethodik und Ökonomie. Deren Argumente werden im folgenden im Detail vorgestellt und anhand der verfügbaren Forschungsliteratur diskutiert.

Sicherheit und Softwaretechnologie

Im Laufe der letzten Jahre ist in Fachbüchern zur Software-/IT-Sicherheit das eine oder andere Mal auch Open-Source angerissen worden,[43] allerdings eher anekdotisch. Systematische Untersuchungen und theoretische Analysen sind noch rar gesät. Aussagekräftig erscheinen vor allem drei Aufsätze aus jüngerer Zeit. McCormack (2001) und Payne (2002) wenden sich der Frage von Qualität resp. Sicherheit von Open-Source empirisch zu, während Challet und Le Du (2003) ein theoretisches Modell zur Fehlerdynamik in der Closed-Source- und Open-Source-Entwicklungsmethode vorstellen. Die Erfahrungen, Argumente und Schlussfolgerungen der Autoren werden hier in aller Kürze referiert.

Alan McCormack hat sich zusammen mit Marco Iansiti und Roberto Verganti zwei Jahre lang in einer empirischen Feldstudie mit den Erfolgen oder Misserfolgen von Softwareprojekten und den unterschiedlichen Ansätze zu ihrer Durchführung befasst (McCormack 2001). Dabei lag der Fokus nicht auf der Frage der Code-Offenheit oder -Geschlossenheit, sondern darauf, ob klassische Entwicklungsmodelle (à la Wasserfall) oder modernere, evolutionäre Modelle erfolgversprechender sind.[44] Dazu wurden 29 abgeschlossene Softwareprojekte (durchgeführt von 17 Firmen) analysiert und die Faktoren herausgearbeitet, die am meisten zu einem erfolgreichen Projektabschluss[45] beigetragen haben (McCormack 2001, S 79). Vier kritische Erfolgsfaktoren konnten identifiziert werden:

- „Early Release of the Evolving Product Design to Customers;“ (S. 79)
- „Daily Incorporation of New Software Code and Rapid Feedback on Design Changes;“ (S. 81)

---

43 Vgl. Schneier (2000, S. 363–365), Viega und McGraw (2002, S. 75–89) und Anderson (2001, S. 536–537).

44 Zwar sind evolutionäre Entwicklungsmodelle in der Softwaretechnik seit Jahrzehnten bekannt, doch werden die meisten Projekte nicht nach dieser Methode entwickelt. Ein schwerwiegender Grund dafür dürfte die Organisation der Software-Entwicklung gemäß einem Prinzipal-Agent-Schema sein: Es gibt eine klare, marktbedingte Trennung zwischen Auftraggeber/Konsument und Hersteller/Lieferant einer Software. Damit verbunden sind die üblichen Probleme des Prinzipal-Agent-Schemas, die aus den mit einer solchen Trennung unvermeidlich einhergehenden Informationsasymmetrien resultieren. Grundlegende Darstellungen der Theorie findet sich in Laffont und Martimort (2002) und Macho-Stadler und Pérez-Castrillo (2001). Eine Anwendung der Theorie auf Software und die Konsequenzen für die Sicherheit findet sich in Gehring (2003). Siehe auch den Abschnitt Sicherheit und Software-Ökonomie des vorliegenden Aufsatzes, der die Argumente noch einmal zusammenfasst.

45 Maßgeblich für die Beurteilung des Projekterfolges waren die Einschätzungen einer Runde von 14 unabhängigen Industriexperten, die in einem 2-Runden-Delphi-Verfahren über Parameter wie „reliability, technical performance (such as speed) and breadth of functionality“ zu befinden hatten (McCormack 2001, S. 79).

- „A Team With Broad-Based Experience of Shipping Multiple Projects;" (S. 81)
- „Major Investments in the Design of the Product Architecture." (S. 82)

Das Ergebnis der Untersuchung lässt keinen Zweifel an der Überlegenheit evolutionärer Entwicklungsmethoden stehen:

> Successful development was evolutionary in nature. Companies first would release a low-functionality version of a product to selected customers at a very early stage of development. Thereafter work would proceed in an iterative fashion, with the design allowed to evolve in response to the customers' feedback. (McCormack 2001, S. 76)

Je weniger Funktionalität im ersten Release der Software, das den Testern zur Verfügung stand, enthalten war, desto höher fiel die Qualität des Endproduktes aus (McCormack 2001, S. 80). Ausschlaggebend ist nach McCormack beim „feedback" nicht die Anzahl der Tester (McCormack 2001, S. 80), sondern die Geschwindigkeit mit der „feedback" auf Änderungen in der Software erfolgt (McCormack 2001, S. 81). Was Raymond als wesentliches Erfolgskriterium der Open-Source-Debugging-Methode beschreibt, scheint tatsächlich als qualitätsrelevantes Paradigma gültig zu sein: „Release early. Release often. And listen to your customers!" (1999, S. 39)

McCormacks Befunde unterstreichen das und er selbst scheint das direkt zu bestätigen, indem er den Linux-Kernel als ein erfolgreiches evolutionäres Projekt anführt (2001, S. 83).

Die Untersuchung von Payne (2002) konzentrierte sich darauf herauszufinden, inwiefern der Open-Source-Prozess geeignet sei, die Sicherheit von Betriebssystemen zu gewährleisten. Faktoren die nach Payne (2002) dafür sprechen könnten sind „[p]eer review" (S. 63), „[f]lexibility and freedom" (S. 65) und andere „miscellaneous arguments" (S. 66). Nach einer kritischen Würdigung auch der Argumente contra Sicherheit bei Open-Source, werden die Ergebnisse einer Studie von 1999 vorgestellt, die den Entstehungsprozess dreier UNIX-artiger Betriebssysteme untersucht hat: Solaris als proprietäres System, Debian GNU/Linux und OpenBSD als Open-Source-Systeme (S. 72). Im Mittelpunkt der Beobachtungen standen die vier „security ‚dimensions': confidentiality, integrity, availability and audit" (S. 72). Denen wurde nach einer in einem früheren Aufsatz (Payne 1999) erläuterten Metrik numerische Werte zugeordnet, die eine Vergleichbarkeit der einzelnen Systemeigenschaften gewährleisten. So kommen die Betriebssysteme hinsichtlich der oben auf-

geführten Sicherheitsdimensionen bei den Sicherheitseigenschaften[46] („features"; S. 73) auf folgende Werte (größere Zahlen stehen für bessere Werte):

- Debian: 6,42; Solaris: 5,92; OpenBSD: 7,03

Man erkennt einen leichten Vorteil beim Open-Source-System Debian ggü. dem proprietären Solaris und einen deutlicheren Vorsprung von OpenBSD ggü. beiden Systemen. Diesen theoretisch ermittelten Werten wurden dann die tatsächlich festgestellten Schwachstellen („vulnerabilities"; S. 73) zur Seite gestellt und analog bewertet. Folgende Werte wurden dabei ermittelt (kleinere Zahlen stehen für geringere Angreifbarkeit/höhere Sicherheit):

- Debian: 7,72; Solaris: 7,74; OpenBSD: 4,19

Während Debian und Solaris hinsichtlich ihrer Schwachstellen praktisch gleichauf liegen, weist OpenBSD deutlich weniger Schwachpunkte auf. Payne (2002) setzt die Werte aus „features" und „vulnerabilites" dann für jedes Betriebssystem zueinander ins Verhältnis und ermittelt einen „Final Score" (S. 73) für jedes System (größere Zahlen stehen für eine bessere Bewertung):

- Debian: -1,0; Solaris: -3,5; OpenBSD: 10,2

Die beiden Open-Source-Betriebssysteme schneiden besser als das proprietäre Solaris ab, wobei der Vorsprung von Debian nicht wirklich überzeugend ausfällt. Nicht zu übersehen ist jedoch der Unterschied zwischen Solaris/Debian auf der einen und OpenBSD auf der anderen Seite. Der Autor schlussfolgert zu Recht, dass „the significant differences between Debian and OpenBSD's score support the argument that making a program ‚open source' does not, by itself, automatically improve the security of the program" (S. 76). Was dann, fragt der Autor weiter, erklärt den großen Unterschied zwischen Debian und OpenBSD? Payne beantwortet die Frage folgendermaßen:

> [W]hile the source code for the Debian system is available for anyone who cares to examine it, the OpenBSD source code is regularly and purposefully examined with the explicit intention of finding and fixing security holes (Payne, 1999), (Payne, 2000). Thus it is this auditing work, rather than simply the general availability of source code, that is responsible for OpenBSD's low number of security problems. This point bears repeating: software will not become automatically more secure by virtue of its source code being published. (2002, S. 76)

Zwar bietet der offen liegende Quellcode bei Open-Source-Software die Voraussetzung, um Software sicherer zu machen, aber die Gestaltung des Entwicklungsprozesses ist entscheidend für die Erfüllung der Erwartungen! Nur, wenn von

[46] Payne (2002, S. 72) erläutert, wie dabei vorgegangen wurde: „Confidentiality features studied included mechanisms such as cryptographic file systems or other data encryption systems. Some of the operating systems considered included the ability to prevent processes from modifying files in certain ways, even if the process were executing with superuser privileges and this acted as an integrity-based security feature. The systems also included security controls specifically relating to the availability dimension which allowed the amount of resources utilized by each process to be specified and limited. The objective here is to prevent a malicious user from over-utilizing a shared resource and denying other users access. Different logging mechanisms were often included such as special software to log network packets received by the computer or to log information about the system call requests made by processes to the kernel. These are examples of some of the availability features included in the systems studied."

der Möglichkeit des Auditing in der Praxis auch Gebrauch gemacht wird, kann der Code sicherer gemacht werden. Offener Code allein ist keine hinreichende Bedingung, wenn auch eine erfolgversprechende Voraussetzung.

Jedwede Diskussion um das Pro und Contra der Sicherheit von Software müsste in diesem Sinne weniger auf das Produkt fokussiert werden als auf den Prozess seiner Herstellung und Distribution. Der „magische Blick" in den Quellcode ist allein kein Garant für einen qualitäts- und sicherheitsbewussten Prozess.[47]

Den Entwicklungsprozess von Software haben Challet und Le Du (2003) in einem theoretischen Modell zur Beschreibung der Fehlerdynamik beschrieben.[48] Ziel ihrer Modellierung war es, unter Annahme typischer Randbedingungen in offenen und geschlossenen Softwareentwicklungsprozessen (Programmierer machen immer Fehler; es gibt nur wenige „geniale" Programmierer, die meisten sind nur mittelmäßig gut; unterschiedliche Programmierer steuern unterschiedlich viel Code zu einem Projekt bei; neuer Code bringt neue Fehler ein; Anwender können potentiell Fehler an die Entwickler melden; der Betreuer eines Projekts entscheidet, welcher Code neu integriert wird; Anwender modifizieren, bei Open-Source, den Code z.T. selbst, um Fehler zu beseitigen) die Entwicklung der Fehlerdynamik, d.h. die phasenweise Zu- und Abnahme von Fehlern im Projektcode, abzubilden. Die Richtigkeit ihres Modells zu Grundegelegt,[49] kommen sie hinsichtlich Closed-Source-Software zu folgender Einsicht:

> This figure already shows that closed source projects are always slower to converge to a bug-free state than open source projects at constant parameters. In addition, ignoring bug reports on already modified code is the best option for closed source projects; this even outperforms open source at short time scales, because the programmers only work on fully buggy parts, hence the bug fixing rate is higher. (S. 3)

---

47 Daher greift auch jede Kritik an Open Source zu kurz, die sich im Kern mit dem „magischen Blick" auseinandersetzt – wie auch jede Befürwortung mit diesem Argument. Insofern ist Oppliger (2003) zuzustimmen. Inwiefern allerdings die von ihm angeführten Randbedingungen zur positiven Beeinflussung der Sicherheit von Software, nämlich (1) ein vollständiger, konsistenter Entwurf; (2) gut (in Sicherheit) geschulte Softwareentwickler; (3) qualitativ hochwertiger Software zur Erstellung; (4) umfassende Tests der erstellten Software; (5) besondere Beachtung der Benutzerschnittstelle; und (6) professionelles, institutionalisierten Patch-Management notwendig seien (S. 675), tatsächlich in ihrer Gesamtheit zutreffen, wird nicht nachgewiesen. Die Arbeiten anderer Autoren lassen jedenfalls Zweifel zu. Die Erkenntnisse von McCormack (2001) etwa sprechen zumindest gegen (1). Und ob bei den am weitesten verbreiteten proprietären Systemen von professionellem Patch-Management gesprochen werden kann, wird jeder mit Skepsis beurteilen, der einmal versucht hat, einen Microsoft-Server auf dem aktuellen Stand zu halten, vgl. z.B. Forno (2003). Jedenfalls lassen sich die von Oppliger angeführten Kriterien nach Auffassung des Autors des vorliegenden Beitrages nicht so zusammfassen, dass „die Diskussionen um Open-Source-Software und entsprechende Lizenzierungsmodelle [...] nicht (sinnvoll) geführt werden können" (Oppliger 2003, S. 675). Zumal wenn man berücksichtigt, dass viele der hier angeführten Quellen, z.B. McCormack (2001) oder Franke und von Hippel (2003), nicht erkenntlich ausgewertet wurden und Fragen der Softwareökonomie weitestgehend unbeachtet blieben.

48 Zur Praxis der Qualitätssicherung in Open-Source-Projekten siehe Zao und Elbaum (2003).

49 Der Aufsatz von Challet und Le Du (2003) wurde zum *peer review* eingereicht, die Aussagen hier stammen alle aus der Vorabversion, wie im Literaturverzeichnis angegeben.

Stimmt diese aus dem theoretischen Modell gewonnene Erkenntnis, könnte sie erklären, warum, bei den kurzen Produktzyklen, die PC-Software heute aufweist, die Hersteller so wenig Energie in die Beseitigung von Fehlern investieren – jedenfalls für den außenstehenden Beobachter.

Nach Durchspielen ihres Modelles mit verschiedenen Parametern ziehen Challet und Le Du (2003) eine Schlussfolgerung, die aus der Perspektive der Qualitätssicherung bei Software bemerkenswert ist:

> From our model we conclude that open source projects converge always faster to a bug-free state for the same set of parameters [...] This finding clearly indicates that closed-source projects must have enough programmers, good enough ones, and have enough users in order to achieve the same software quality in the same amount of time. On the other hand, the quality of open source project programmers does not need to be as high as those of close source projects in order to achieve the same rate of convergence to bug-free programs. (S. 6–7)

Man muss kein gläubiger Anhänger der Open-Source-Community sein, um die grosse Tragweite einer solchen Aussage zu ermessen. Zieht man die Befunde zur Qualifikation von Open-Source-Entwicklern heran, wie sie Studien wie Robles, Scheider, Tretkowski und Weber (2001), FLOSS (2002) und Lakhani, Wolf, Bates und DiBona (2002) geliefert haben, scheint es mehr als naheliegend zu sein, dass Open-Source-Projekte auch in der Praxis die fehlerfreiere Software hervorzubringen vermögen. Unter Berücksichtigung des diskutierten Zusammenhanges zwischen Fehlern von Software und daraus resultierenden Verletzlichkeiten, bedeutet das, dass Software aus Open-Source-Projekten weniger verletzlich ist.[50]

Sicherheit und Software-Ökonomie

Dass die Sicherheit von Software abhängig ist von den ökonomischen Randbedingen ihrer Entstehung und Verbreitung sowie ihres Einsatzes, ist in der Wissenschaft mittlerweile erkannt und akzeptiert.[51]

Ross Anderson, bekannter Fachmann für Sicherheit an der Universität Cambridge in England, stellte unlängst fest, „Open and Closed Systems are Equivalent" (2003). Allerdings schränkt er seine Feststellung ein und lässt sie *so* nur für „an ideal world" gelten. In einem theoretischen Modell zum Wachstum der Systemzuverlässigkeit legt er dar, dass beide Ansätze (Closed Sourceund Open-Source) vergleichbar schnell (oder langsam) an Zuverlässigkeit zunehemende Software hervorbringen

[50] Ausnahmen bestätigen auch hier wieder die Regel. Empirische Befunde jedenfalls stützen diese Aussage. So kann man z.B. anhand der verfügbaren CERT-Informationen zeigen, dass bei einem Vergleich zweier funktional ähnlicher Produkte aus Closed-Source- und aus Open-Source-Produktion, Microsofts Webserver IIS und Open-Source-Webserver Apache, das Open-Source-Produkt qualitativ weniger verletzlich, also sicherer ist (Gehring 2003a). Das gelegentlich gegen einen solchen Vergleich vorgebrachte Argument der geringeren Verbreitung von Open-Source-Produkten greift hier nicht: der IIS hat ca. 23.5%, der Apache ca. 69% Marktanteil (laut http://news.netcraft.com/archives/2004/01/01/january_2004_web_server_survey.html, 8.1. 2004).

[51] Vgl. etwa die Beiträge zu den zwei Workshops on Economics and Information Security (WEIS 2002; WEIS 2003).

würden. In der wirklichen Welt allerdings würde eine Reihe externer Einflussfaktoren insbesondere ökonomischer Art dazu führen, dass die thoretisch erreichbare Äquivalenz des Zuverlässigkeitswachstums in der einen oder anderen Richtung aus der Balance geschoben würde, abhängig vom jeweils betrachteten Parameter. Zur Beurteilung dessen, wie denn in der Praxis sich die Waage zugunsten des einen oder anderen Entwicklungs- und Distributionsmodelles neigen würde, fordert er mehr empirische Untersuchungen,[52] worin ihm sicher Recht zu geben ist. Zwar gibt es eine Reihe empirischer Studien, die zugunsten von Open-Source zu sprechen scheinen. Es dürfte jedoch geboten sein, Vorsicht walten zu lassen, wenn daraus generalisiert werden soll. Zu wenig repräsentativ, zu stark auf einzelne Projekte fokussiert sind die vorliegenden Untersuchungen bisher.

Der Beitrag von Landwehr (2002) hebt hervor, dass aus Sicherheitsgründen ein besserer Informationsfluss notwendig sei, um die Dysfunktionalitäten des Marktes für „information security" zu überwinden. Damit bringt er ein Argument vor, das analog in Gehring (2001; 2003) zu finden ist, was das Problem der asymmetrischen Informationen (s.u.) und seine Auswirkungen betrifft. Auch Landwehr ist der Auffassung, dass quelloffene Software einen Beitrag dazu leisten kann, den Anwendern die rationale Entscheidung *pro* oder *contra* eine bestimmte Software zu erleichtern. Nur so kann man letztlich dem Problem der adversen Auslese (siehe FN 57) entgegenwirken.

In Gehring (2003) stellt der Autor des vorliegenden Beitrages die Frage der Software-Ökonomie insbesondere in den Zusammenhang mit Schutzrechten wie Urheber- und Patentrecht.[53] Da Software sowohl für den Urheberrechtsschutz (auf der textuellen und strukturellen Ebene) als auch für den Patentschutz[54] (auf der funktionalen Ebene) in Frage kommt, müssen die daraus resultierenden ökonomischen Konsequenzen Berücksichtigung finden, wenn es um Sicherheitsfragen geht. Die Tatsache, dass es sich bei Rechten des „geistigen Eigentums" um exklusive Schutzrechte mit sehr weitem Schutzumfang und einer massiven Störwirkung auf den Wettbewerb[55] handelt, garantiert großen Anbietern im Markt potentiell hohe Profite. Dass diese Profite in nicht unerheblichem Umfang realisiert werden, zeigt das Beispiel Microsoft. Auf der anderen Seite kennen weder das Urheber- noch das Patentrecht einen spezifischen Fehlerbegriff und damit verbundene Haftungsregelungen. In der Praxis wirkt sich das so aus, dass Softwarehersteller nur in sehr wenigen Ausnahmefällen für ihre fehlerhaften Produkte in Haftung genommen werden können.

---

52 Instruktiv ist diesbezüglich Wheeler (2003).

53 Erste Betrachtungen dazu wurden in Gehring (2001) vorgestellt. Die ökonomische Struktur des Rechts des „geistigen Eigentums" wird umfassend analysiert in Landes und Posner (2003), die allerdings Software nur am Rande behandeln.

54 Eine umfangreiche Darstellung findet sich bei Stobbs (2000).

55 Schutzrechte für „geistiges Eigentum" blockieren den Imitationswettbewerb, weshalb in der ökonomischen Literatur regelmäßig von Monopolrechten die Rede ist. Vgl. z.B. Landes und Posner (2003, S. 15), die allerdings zurecht darauf hinweisen, dass eine Gleichsetzung von rechtlichem und ökonomischem Monopol in der Regel nicht angebracht ist, wenn der Inhaber des rechtlichen Monopols nicht auch gleichzeitig über große Marktmacht verfügt.

Aus dieser Analyse wird geschlussfolgert, dass Softwarehersteller einerseits große (Profit-) Anreize haben, Software zu produzieren und zu vermarkten, andererseits aber wenig Anreize, (nur) fehlerfreie, sichere Software zu vermarkten. Informationsasymmetrien zwischen Softwareanbieter (bzw. Hersteller) und Softwarenutzer tragen ihren Teil dazu bei, dass ein Markt für sichere Software nicht entstehen kann, der den Bedürfnissen einer global vernetzten IT-Landschaft genügen würde. „Adverse selection“[56] sorgt dafür, dass sich sichere Produkte nicht in der Breite etablieren können, die für ein signifikant höheres Sicherheitsniveau notwendig wäre. Intellectual Property Rights verschärfen das Problem insofern, als dass Selbsthilfe, z.B. in Form eines „reverse compilation“[57] zu Evaluationszwecken, rechtlich weitgehend für unzulässig erklärt worden ist.

Im Anschluss an diese Feststellung fragt Gehring (2003), inwiefern Open-Source-Software geeignet sein könnte, Abhilfe zu schaffen. Zwar löst Open-Source-Software nicht das Problem der unzureichenden Haftung bei Software, aber sie beseitigt in erheblichem Maße die Informationsasymmetrien zwischen Hersteller und Anwender:

- Anwender haben Einblick in den Quellcode, erfahren also, was genau sie erwerben;
- Sicherheitsmängel können im Quellcode evaluiert werden, ggf. als Dienstleistung durch Dritte.

Der Quellcode bietet damit eine Signalfunktion,[58] wie sie zur Überwindung von Informationsasymmtrien dringend benötigt wird: Wenn ein Anwender sich vor einem geplanten Einsatz überzeugen will, ob ein proprietäres Produkt oder ein Open-Source-Produkt vergleichbarer Funktionalität seinen Zwecken besser genügt, ist er zu einer Evaluation der Produkteigenschaften gezwungen. Zur Evaluation stehen für den Anwender unterschiedliche Signale zur Auswertung bereit: (1) Werbeinfor-

[56] Das Problem der adversen Auslese (adverse selection) entsteht in Märkten, in denen die potentiellen Vertragspartner vor Vertragsschluss über asymmetrische Informationen verfügen: Einer der beiden potentiellen Vertragspartner, der Anbieter eines Produkts oder einer Leistung, weiss typischerweise mehr über die Qualität seines Angebots als der potentielle Käufer, dem die Beziehung zwischen Qualität und Preis vor dem Erwerb und Ausprobieren des Angebots nicht objektiv klar ist. Konfrontiert mit anscheinend vergleichbaren Angeboten und unterschiedlichen Preisen wird er maximale Leistung bei minimalen Kosten anstreben und sich für das preiswerteste Angebot entscheiden. Wenn Anbieter höherer Qualitäten nicht in der Lage sind, auf Grund ihrer Selbstkosten, vergleichbare Preise anzubieten, werden sie nicht wettbewerbsfähig sein. Eine Lösung besteht für dieses Dilemma dann darin, die Qualität des Angebots zu reduzieren, um ebenfalls niedrige Preise anbieten zu können. Auf diese Weise sinkt die Menge der im Markt angebotenen Qualitätsprodukte und nur die „schlechten Qualitäten“ werden gehandelt. Eine Lösungsmöglichkeit ist z.T. das *signalling*, wie im Text diskutiert.

[57] Unter „reverse compilation“ versteht man die Rekonstruktion der programmsprachlichen Präsentation eines binären Programmes in einer für den Menschen lesbaren Form. „Reverse compilation“ ist nicht zu verwechseln mit „reverse engineering“, bei dem es im Kern um die Sichtbarmachung der zugrundeliegenden Ideen und Konzepte einers Programmes geht. Allerdings setzt „reverse engineering“ in der Regel eine „reverse compilation“ voraus, sodass die Grenzen fließend sind.

[58] Mithilfe solcher Funktionen werden dem potentiellen Kunden Qualitätsmerkmale, bzw. allgemein Produkteigenschaften signalisiert, die nicht ohne weiteres beobachtet werden können. So signalisiert etwa eine freiwillige Garantie das Vertrauen eines Anbieters in die Haltbarkeit seines Produkts und eine Marke überträgt Erfahrungen aus der Vergangenheit auf Erwartungen an die Zukunft; Patente können zur Signalisierung der Innovativität eines Anbieters dienen, usw.

mationen des Anbieters; (2) Befragung von Experten (oft indirekt, via Artikel in der Fachpresse); (3) Ausprobieren des Produkts; (4) Begutachtung der „inneren Werte", d.h. Aufbau, Ausführung, Dokumentation usw. des Programmcodes.

Vergleicht man hinsichtlich dieser Optionen proprietäre (Closed-Source-) Software und Open-Source-Software, so kann man zu folgender Gegenüberstellung[59] gelangen:

Informationsbeschaffung	Closed-Source	Open-Source
aus Werbung	ja	ja (zum Teil)
Expertenmeinungen	ja	ja
Erfahrung durch Ausprobieren	nein	ja
eigene Auditierung	nein	ja
unabhängige Auditierung	selten	ja

Informationsbeschaffung vor dem Erwerb/Einsatz

Die Informationsasymmetrien zwischen Anbieter und Anwender (Kunde) können mithin auf Grund der Lizenzbedingungen rechtlich und, dank des einsehbaren Quellcodes bei Open-Source-Software, auch faktisch weitestgehend abgebaut werden. Gerade der Faktor Sicherheit lässt sich nur sehr schlecht[60] bewerten, wenn der Quellcode nicht zur Verfügung steht.[61] Gerade hier kommt der Signalfunktion (4), d.h. den offenliegenden Quellcodes bei Open-Source-Software, immense Bedeutung zu.

Erst mit der Beseitigung der Informationsasymmetrien kann ein echter Qualitätswettbewerb[62] entstehen, der zu dem gewünschten Ergebnis führt: qualitativ bes-

59 Ausnahmen für Großkunden, Regierungen usw. sind zwar bei proprietärer Closed-Source-Software im Einzelfall hilfreich, im Normalfall jedoch unerreichbar.

60 Im Prinzip kann bei nicht vorliegendem Quellcode nur eine quantitative Bewertung durch massive Penetrationstests („Zerstörungsprüfung") erfolgen, die statistische Aussagen über die Produktsicherheit zulässt.

61 Aus diesem Grunde ist bei einer Sicherheitszertifizierung nach *Common Criteria (CC)* oder *Information Technology Security Evaluation Criteria (ITSEC)* in den höheren Evaluationsstufen die Vorlage des Quellcodes unverzichtbar.

62 Der Wettbewerb wird zwar im Markt für Produkte, aber nicht zwischen klassischen Firmen, sondern zwischen Open-Source-Projekten und anderen Open-Source-Projekten bzw. Firmen ausgetragen. Aus der Perspektive eines Marktes in dem dominierende Anbieter von wettbewerbswidrigem Verhalten nicht effektiv abgehalten werden können (vgl. den Ausgang des Microsoft-Prozesses und Microsofts anschließendes Verhalten), ist dieser Umstand von großer Bedeutung. Im Unterschied zu einer Firma kann ein Projekt (das unterschiedlichste Institutionen einschließt, darunter auch Firmen) nicht ohne weiteres aus dem Markt verdrängt werden, etwa durch Aufkauf oder kostspielige Gerichtsverfahren. Potentielle Wettbewerber im Markt für Produkte greifen stattdessen die rechtliche Basis der Projekte an (siehe den Prozess SCO vs. IBM in dem die Copyright-Konformität der GNU Public License in Frage gestellt wird), setzten Patente als „strategische Waffen" ein (etwa in Standardisierungsprozessen), installieren proprietäre Schnittstellenformate die eine Interoperabilität von Open-Source-Produkten mit proprietären Produkten erschweren oder versuchen, die offene PC-Plattform gegen Modifikationen von Seiten des Computerbesitzers abzusichern (sog. „trusted platforms"), was sich negativ auf Open Source und andere Wettbewerber auswirken könnte

sere (sicherere) Produkte setzen sich durch. Ausgehend von dieser Einsicht schlägt Gehring (2003) zur Erhöhung der IT-Sicherheit ein Risikomanagement vor, dessen Fundament Open-Source-Software sein sollte. Darauf sollte ein umfassendes System des Source-Code-Auditing aufgebaut werden, das z.B. bei Beschaffungsentscheidungen der Verwaltung eine marktstimulierende Rolle übernehmen könnte. Insofern Bestimmungen zum Schutz des „geistigen Eigentums" (etwa im Bereich des „reverse engineering" usw.) den Sicherheitsbedürfnissen entgegenstehen, sollten sie geeignet modifiziert werden.

Franke und von Hippel (2003) haben einen anderen Problembereich des klassischen Softwaremarktes untersucht: Nutzerbedürfnisse sind, in Abhängigkeit von der Art der Produkte, in unterschiedlichem Umfang heterogen. Ein Anbieter muss diese Heterogenität erforschen und in seiner Produktstrategie berücksichtigen. Dazu werden beispielsweise Märkte segementiert und je Segment die angebotenen Produkte in ihren Eigenschaften und Preisen differenziert.[63] Sollte es aus Anbietersicht zu kostspielig sein, bestimmte Marktsegmente zu bedienen, werden dafür keine passenden Produkte angeboten (Franke und von Hippel 2003, S. 1199). Die potentiellen Kunden können ihre Nachfrage dann nicht über den Markt decken.

Mit dem Open-Source-Modell ist es nun möglich, einen Teil der Entwicklungskosten für einen Markt mit sehr heterogenen Bedürfnissen auf die Anwender zu übertragen, wodurch erheblich mehr Marksegmente bedient werden können. Das trifft auch auf die Sicherheitseigenschaften von Softwareprodukten zu (S. 1200). Diese Überlegungen wurden von Franke und von Hippel (2003) am Beispiel des Apache-Webservers empirisch überprüft. Die Befunde ergaben,

> [...] that user needs for Apache security functionality are in fact highly heterogeneous. Next, we find that many Apache users are not fully satisfied by existing standard Apache security offerings. When we apply a very conservative measure of willingness to pay (WTP) (an 80% deflation of expressed willingness to pay), we find that the average Apache user is willing to pay a considerable amount (over US$ 5000 per user and over US$ 160 million in aggregate) to have their individual needs for the security functions of Apache met to their total satisfaction. (S. 1200)

Die Tatsache, dass der Source Code des Apache-Webservers als Open-Source den Anwendern zur Verfügung steht, ermöglicht es ihnen, je nach Investitionsbereitschaft, in die Verbesserung der Sicherheit zu investieren. Ein Teil der Anwender setzt die Möglichkeit in die Tat um und modifiziert den Apache-Code mit dem Ergebnis, dass ihre Sicherheitsanforderungen besser befriedigt werden können:

(Anderson 2003; Koenig und Neumann 2003). Doch besteht da keine Zwangsläufigkeit; vielmehr bieten „trusted platforms" gerade auch in Verbindung mit dem Open-Source-Modell ein großes Potential, IT-Systeme sicherer zu machen, ohne den Wettbewerb auszuschließen (Kuhlmann und Gehring 2003, S. 200–202).

63 Das einfachste Beispiel ergibt sich aus den unterschiedlichen Zahlungspräferenzen potentieller Autokäufer. Für Kunden mit niederiger Kaufbereitkraft werden einfachere Produkte (Automodelle) angeboten, für Kunden mit hoher Kaufkraft hochwertige Produkte (Luxusautos). In der Grundfunktionalität unterscheiden sich die Produkte jedoch nicht wesentlich: „Auto ist Auto!" (und alle fahren sie auf Rädern). Mit der Qualitäts-/Preisdifferenzierung kann der Anbieter einen wesentlich größeren Kundenkreis bedienen und dadurch höhere Gesamtprofite erwirtschaften.

> When we compare responses from innovating and non-innovating users, we find that users that modify the standard product report significantly higher satisfaction levels than those that do not. (S. 1200)

Hinzu kommt, dass auch Anwender mit geringerer (oder ohne) Investitionsbereitschaft von den Modifikationen profitieren, wie es bei der Open-Source-Methode durch die Weitergabe des verbesserten Quellcodes die Regel ist.[64] Unter Berücksichtigung der Heterogenität anderer Märkte, die der im Apache-Fall vergleichbar ist, schließen Franke und von Hippel, dass der Einsatz von „toolkits for user innovation," wie beispielsweise die Zurverfügungstellung von Software im Quellcode und entsprechende Lizenzierung, dazu führen könne, die Nutzerbedürfnisse – auch ihre Sicherheitsbedürfnisse – besser zu befriedigen als durch klassische Produktentwicklung und -vermarktung (2003, S. 1200).

Zusammenfassung und Schlussfolgerung

Im vorliegenden Aufsatz wurde versucht, die vier wesentlichen Pfeiler der bisherigen Debatte um die Sicherheit von Open-Source und ihrer Hintergründe zu identifizieren, sowie auf ein erhebliches Defizit im Bereich der Psychologie und Soziologie der Softwareentwicklung hinzuweisen.

Dabei wurde deutlich gemacht, dass die Debatte zum Teil zwar auch auf polemischen Elementen aufsetzt, zum überwiegenden Teil jedoch (aus wissenschaftlicher Sicht) gerechtfertigt erscheint.

Zwei Pfeiler der Debatte, nämlich:

- der regelmäßig behauptete Zusammenhang zwischen dem quelloffenen Prozess der Softwareerstellung/-distribution von Software aus technischer Sicht sowie

[64] Dieser Befund stützt eine Vermutung des Autors, dass „free riding", also Trittbrettfahrerei, in der ökonomischen Literatur immer als problematisch angesehen (vgl. Fritsch, Wein und Ewers 2001, S. 90 ff.; Macho-Stadler und Pérez-Castrillo 2001, S. 90; Varian 2002), in Netzwerkumgebungen in Maßen durchaus wünschenswert sein kann, wo es um die Sicherheit des Netzwerkes geht (zumal wenn, wie im Apache-Beispiel, ganz offensichtlich individuelle Investitionsanreize vorhanden sind, obwohl das zu erwartende „free riding" ja vorher bekannt ist – aus den Apache-Lizenzbedingungen). Aus demselben Grund ist es sinnvoll, zur Bekämpfung einer Epidemie Medikamente kostenlos abzugeben, wenn Externalitäten existieren (akute Ansteckungsgefahr). Ein weiteres ökonomisches Argument hat Dirk Kuhlmann vorgebracht: Angenommen, ein Anwender eines bestimmten Open-Source-Programmes stellt einen Fehler fest, sucht und findet den Fehler im Quellcode und beseitigt ihn. Handelt es sich um einen sicherheitsrelevanten Fehler, dessen Auftreten auf anderen Systemen ihm selbst einen Vorteil verschaffen kann, steht er vor einem Dilemma: Soll er den Fehler und dessen Beseitigung („patch") an die Entwickler weitermelden? Tut er das, verliert er auf der einen Seite den erwähnten Vorteil; auf der anderen Seite wird er, vorausgesetzt, die Entwickler beseitigen den Fehler ein für alle Mal, in Zukunft bei neuen Programmversionen selbst keinen Beseitigungsaufwand (d.h. Kosten) mehr haben. Hinzu kommt, dass andere Anwender den Fehler ihrerseits finden und melden könnten, d.h. sein Vorteil ohnehin nur kurz währen würde. „Unter der Annahme, dass die meisten Benutzer ehrliche Leute sind und es für sie billiger ist, den Bug zu melden und damit das Problem ein für alle Mal aus der Welt zu schaffen, gibt es also einen ökonomischen Anreiz zur Kooperation im IT-Sicherheitsbereich. Das ist eine gute Sache, denn fehlende Kooperation ist eines der Hauptprobleme in dieser Sparte." (persönliche Kommunikation mit D. Kuhlmann 2004)

- die ökonomischen Randbedingungen einer klassischen, marktwirtschaftlichen Organisation der Softwareherstellung/-distribution vs. der Open-Source-Methode

und ihre Einflüsse auf Qualität/Sicherheit der resultierenden Softwareprodukte wurden anhand der aktuellen wissenschaftlichen Literatur diskutiert. Sowohl die vorgestellten Befunde aus empirischen Untersuchungen als auch die theoretischen Überlegungen lassen es in ihrer Gesamtheit als gerechtfertigt erscheinen, die Frage danach, ob denn Open-Source bessere – und sicherere – Software hervorbringen würde, mit *„ja, oft"* zu beantworten.

Eine solche Erkenntnis müsste in Anbetracht der durch fehlerhafte Software verursachten volkswirtschaftlichen Schäden auf politischer Ebene normalerweise zu Schlussfolgerungen führen:

Statt das Closed-Source-Modell bevorzugt zu behandeln, z. B. durch urheber- und patentrechtliche Unterstützung,[65] *müsste die breite Anwendung eines auf dem Open-Source-Modell basierenden Risikomanagements durch die Schaffung geeigneter Anreizstrukturen gefördert werden.*[66]

Jedenfalls müssten gewichtige und prüfbare Gegenargumente ins Feld geführt werden, um Forderungen nach einem solchen Paradigmenwechsel in Anbetracht der mittlerweile vorliegenden wissenschaftlichen Befunde zu deligitimieren. Die Beweislast liegt inzwischen, scheint es, auf Seiten der Befürworter des Closed-Source-Modells.

Literatur

Anderson, Ross J. (2001): *Security Engineering: A Guide to Building Dependable Distributed Systems,* New York: John Wiley & Sons

Anderson, Ross J. (2001a): *Why Information Security is Hard – An Economic Perspective*, in: Proceedings of the Seventeenth Computer Security Applications Conference 2001, Los Alamitos, CA: IEEE Comput. Society, online http://www.cl.cam.ac.uk/ftp/users/rja14/econ.pdf

Anderson, Ross J. (2003): *Open and Closed Systems are Equivalent (that is, in an ideal world)*, online http://www.cl.cam.ac.uk/ftp/users/rja14/toulousebook.pdf

Beck, Kent (2000): *Extreme Programming Explained*, Boston, MA: Addison-Wesley

Bollier, David (2003): *Silent Theft: The Private Plunder of Our Common Wealth*, New York und London: Rouledge

Boyle, James (1996): *Shamans, Software* & *Spleens: Law and the Construction of the Information Society*, Cambridge, MA & London: Harvard University Press

Boehm, Barry W. (1981): *Software Engineering Economics*, Englewood Cliffs, NJ: Prentice Hall

[65] Vgl. Lutterbeck, Horns und Gehring (2000) und Gehring (2001; 2003a).

[66] Dass dabei durchaus im Einzelfall zu differenzieren ist, wird aus den Einsichten von McCormack (2001) und Payne (2002) klar. Und welche Anreizstrukturen erfolgversprechend sein werden, lässt sich nicht mit Gewissheit vorhersagen. Die Bereitstellung von Infrastrukur durch die DARPA im Auftrage des US-Verteidigungsministeriums zum Zwecke „at drawing skilled eyeballs to the thankless task of open-source security auditing" (Poulsen 2004) ist jedenfalls gescheitert.

Branscomb, Anne Wells (1994): *Who Owns Information? From Privacy to Public Access*, New York: Basic Books

Campbell-Kelly, Martin und William Aspray (1996): *Computer: A History of the Information Machine*, New York: Basic Books

Carini, Brian und Benoit Morel (2002): *Dynamics and Equilibria of Information Security Investment*, in: WEIS (2002)

Clapes, Anthony L. (1993): *Softwars: The Legal Battles for Control of the Global Software Industry*, Westport, CT: Quorum Books

Cockburn, Alister (2002): *Agile Software Development*, Boston, MA: Addison-Wesley

Correa, Carlos M. (2000): *Intellectual Property Rights, the WTO and Developing Countries. The TRIPS Agreement and Policy Options*, London und New York: Zed Books

Dignatz, Eitel (1999): *Sicherheit durch Open Source*, Gastkommentar in: Computerwoche vom 13. 08. 1999,
online http://www.dignatz.de/d/spotlight/artikel/oss+sicherheit_19990813_002.html (7.1.2004)

Dorchester, Dave (1999): *Why License Software Engineers?* In: IEEE Software, Vol. 16, No. 2 (March/April 1999), S. 101–102

Drahos, Peter und John Braithwaite (2002): *Information Feudalism: Who Owns the Knowledge Economy?* New York: The New Press

Economist (2003): *Microsoft at the power point*, in: Economist News, 11. September 2003,
online http://www.economist.com/business/displayStory.cfm?story_id=2054746

Fisk, Mike (2002): *Causes & Remedies for Social Acceptance of Network Insecurity*, in: WEIS (2002)

FLOSS (2002): *Free/Libre and Open Source Software: Survey and Study*, Forschungsbericht, International Institute of Infonomics, University of Maastricht, The Netherlands und Berlecon Research GmbH, Berlin, Deutschland, June 2002, http://www.infonomics.nl/FLOSS/report/

Forno, Richard (2003): *Overcoming ‚Security By Good Intentions'*, in: The Register News vom 9. Juni 2003,
online http://www.theregister.co.uk/content/55/31094.html

Franke, Nikolaus und Eric von Hippel (2003): *Satisfying heterogeneous user needs via innovation toolkits: the case of Apache security software*, in: Research Policy, Jg. 32, Nr. 7, S. 1199–1215,
online http://dx.doi.org/10.1016/S0048-7333(03)00049-0; Vorabversion erschienen als MIT Sloan School of Management Working Paper # 4341-02, (January 2002)
online http://web.mit.edu/evhippel/www/ApacheHeteroWP.pdf

Frailey, Dennis J. (1999): *Licensing Software Engineers*, in: Communications of the ACM, Vol. 42, No. 12, S. 29–30

Fritsch, Michael, Thomas Wein und Hans-Jürgen Ewers (2001): *Marktversagen und Wirtschaftspolitik*, 4. Aufl., München: Vahlen

Gehring, Robert A. (2001): *Software Patents – IT-Security at Stake?* Präsentation auf dem Kongress „Innovations for an e-Society. Challenges for Technology As-

sessment“, Berlin, 17.–19. Oktober 2001,
online http://ig.cs.tu-berlin.de/ap/rg/2001-10/index.html.

Gehring, Robert A. (2003): *2003 (1) The Journal of Information, Law and Technology (JILT)*,
online http://elj.warwick.ac.uk/jilt/03-1/gehring.htm

Gehring, Robert A. (2003a): *Open Source Software – Sicherheit im Spannungsfeld von Ökonomie und Politik*, Beitrag auf dem CAST-Workshop „Sicherheit mit Open Source?“, Darmstadt, 20. März 2003, Präsentation und Text via http://ig.cs.tu-berlin.de/ap/rg/

Gehring, Robert A. (2003b): *Business Case: Open Source Security*, Präsentation auf der Euroforum-Jahrestagung „Sicherheit 2003“, Hamburg, 11.–14. November 2003,
online http://ig.cs.tu-berlin.de/ap/rg/2003-11/Gehring-Euroforum-OSSSecurity-112003.pdf.

Goltzsch, Patrick (2003): *Linux beschleunigt seinen Vormarsch*, in: FTD News vom 29. Dezember 2003,
online http://www.ftd.de/tm/it/1072525171182.html?nv=hpm.

Gordon, Kawrence A. und Martin P. Loeb (2002): *The Economics of Information Security Investment*, in: ACM Transactions on Information and System Security, Vol. 5, No. 4 (November 2002), S. 438–457.

Greenwals, Michael, Carl A. Gunter, Björn Knutsson, Andre Scedrov, Jonathan M. Smith und Steve Zdancevic (2003): *Computer Security is Not a Science (but it should be)*,
online http://www.cis.upenn.edu/~stevez/papers/GGKS+03.pdf

heise (2004): *Bill Gates: Microsofts Daten-Armbanduhr auch in Europa*, in: heise newsticker vom 9. Januar 2004,
online http://www.heise.de/newsticker/data/jk-09.01.04-001/

Hillmann, Karl-Heinz (1994): *Wörterbuch der Soziologie*, 4. Aufl., Stuttgart: Alfred Kröner

Hofmann, Jan (2002): *Free software, big business?* Forschungsbericht Nr. 32, Deutsche Bank Research, Frankfurt am Main,
online http://www.dbresearch.de/PROD/PROD0000000000047532.pdf

Hoglund, Greg (2002): Security Band-Aids: *More Cost-Effective than „Secure“ Coding*, in: IEEE Software, Vol. 19, No. 6 (November/December 2002), S. 57–58

Hohmann, Luke (1996): *Journey of the Software Professional: A Sociology of Software Development*, Upper Saddle River, NJ: Prentice Hall PTR

Imparato, Nicholas, Hrsg. (1999): *Capital for Our Time: The Economic, Legal, and Management Challenges of Intellectual Capital*, Stanford, CA: Hoover Institution Press

Ishii, Kei und Bernd Lutterbeck (2002): *Der Microsoft-Prozess*, in: Alexander Roesler und Bernd Stiegler (Hrsg.): *Microsoft: Medien. Macht. Monopol*, S. 130–153, Frankfurt am Main: Suhrkamp,
online http://ig.cs.tu-berlin.de/bl/072/IshiiLutterbeck-MicrosoftProzess-2002.pdf

Jonietz, Erika (2004): *Valid Voring?* In: MIT Technology Review, Jg. 107, 2004, Nr. 1 (Februar), S. 74–75

Jonsson, Erland, Lars Strömberg und Stefan Lindskog (2000): *On the Functional Relation Between Security and Dependability Impairments*, in: Proceedings of the 1999 New Security Paradigm Workshop, September 1999, Ontario, S. 104–111

Kalam, A. P. J. Abdul (2003): *Convergence of Technologies*, Rede für das International Institute of Information Technology, 28. Mai 2003,
online http://presidentofindia.nic.in/S/html/speeches/others/may28_2003_2.htm

Kamerling, Erik (2003): *Three Questions for the October 8, 2003 Top 20 Briefing*,
online http://www.sans.org/top20/overview03.pdf.

Knight, John C. (2002): *Dependability of Embedded Systems*, in: Proceedings of ICSE'02, 19.–25. May 2002, Orlando, Florida, S. 685–686

Koenig, Christian und Andreas Neumann (2003): *Standardisierung und EG-Wettbewerbsrecht – ist bei vertrauenswürdigen Systemumgebungen wettbewerbspolitisches Misstrauen angebracht?* In: WuW 11/2003, S. 1138–1152

Köhntopp, Kristian, Marit Köhntopp und Andreas Pfitzmann (2000): *Sicherheit durch Open Source? Chancen und Grenzen*, in: Datenschutz und Datensicherheit (DuD), Jg. 24, Nr. 9, S. 508–513

Kolawa, Adam (2002): *Certification Will Do More Harm Than Good*, in: IEEE Computer, Vol. 35, No. 6, S. 34–35

Krsul, Ivan, Eugene Spafford und Mahesh Tripunitara (1998): *Computer Vulnerability Analysis*, 6. Mai 1989. Technical Report COAST TR98-07, COAST Laboratory, Purdue University, West Lafayette, IN,
online http://ftp.cerias.purdue.edu/pub/papers/ivan-krsul/krsul9807.pdf

Kuhlmann, Dirk und Robert A. Gehring (2003): *TCPA, DRM, and Beyond*, in: Eberhard Becker, Willms Buhse, Dirk Günnewig und Niels Rump (Hrsg.): Digital Rights Management: Technological, Economic, Legal and Political Aspects, Berlin, Heidelberg, S. 178–205, New York: Springer Verlag

Laffont, Jean-Jacques und David Martimort (2002): *The Theory of Incentives*, Princeton, NJ und Oxford: Princeton University Press

Lakhani, Karim R., Bob Wolf, Jeff Bates und Chris DiBona (2002): *The Boston Consulting Group Hacker Survey*,
online http://www.osdn.com/bcg/bcg-0.73/BCGHackerSurveyv0-73.html

Landes, William M. und Richard A. Posner (2003): *The Economic Structure of Intellectual Property Law*, Cambridge, MA und London: Belknap/Harvard University Press

Landwehr, Carl (2002): *Improving Information Flow in the Information Security Market*, in: WEIS (2002)

Lawton, George (2002): *Open Source Security: Opportunity or Oxymoron?* In: IEEE Computer, Vol. 35, No. 3, S. 18–21

Lemley, Mark A., Peter S. Menell, Robert P. Merges und Pamela Samuelson (2002): *Software and Internet Law*, 2. Aufl., New York: Aspen Publishers

Lessig, Lawrence (1999): *Code and Other Laws of Cyberspace*, New York: Basic Books

Levy, Leon S. (1987): *Taming the Tiger – Software Engineering and Software Economics*, New York: Springer Verlag

Levy, Steven (2001): *Hackers: Heroes of the Computer Revolution*, New York: Penguin Books

Li, Wei (2002): *Security Model of Open Source Software*, online http://www.cs.helsinki.fi/u/campa/teaching/oss/papers/wei.pdf.

Litman, Jessica (2001): *Digital Copyright*, Amherst, N.Y.: Prometheus Books

Luqi und Joseph A. Gogue (1997): *Formal Methods: Promises and Problems*, in: IEEE Software, Vol. 14, Iss. 1, Januar 1997, S. 73–85

Lutterbeck, Bernd, Axel H. Horns und Robert A. Gehring (2000): *Sicherheit in der Informationstechnologie und Patentschutz für Softwareprodukte – Ein Widerspruch?,* Kurzgutachten erstellt im Auftrag des Bundesministeriums für Wirtschaft und Technologie, Berlin, online http://www.sicherheit-im-internet.de/download/Kurzgutachten-Software-patente.pdf

Macho-Stadler, Inés und J. David Pérez-Castrillo (2001): *An Introduction to the Economics of Information: Incentives and Contracts*, Oxford und New York: Oxford University Press

Macrina, Francis L. (1995): *Scientific Integrity: An Introductory Text with Cases*, Washington, D.C.: ASM Press

Maggs, Peter B., John T. Soma und James A. Sprowl (2001): *Internet and Computer Law: Cases-Comments-Questions*, St. Paul, MN: West Group

Meadows, Catherine und John McLean (1999): *Security and Dependability: Then and Now*, in: Proceedings of Computer Security, Dependability, and Assurance: From Needs to Solutions, 7–9 July 1998 and 11–13 November 1998, York, UK & Williamsburg, VA, USA, S. 166–170. Los Alamitos, CA: IEEE Comput. Society

McChesney, Robert W., Ellen Meiksins Wood und John Bellamy Foster (1998): *Capitalism and the Information Age: The Political Economy of the Global Communication Revolution*, New York: Monthly Review Press

Mundie, Craig (2003): *Security: Source Access and the Software Ecosystem*, online http://www.microsoft.com/resources/sharedsource/Security/SourceAccess.mspx

Mustonen, Mikko (2003): *Copyleft – the economics of Linux and other open source software*, in: Information Economics and Policy, Vol. 15, S. 99–121

Myers, Glenford J. (1976): *Software Reliability: Principles and Practice*, New York: John Wiley & Sons

Myers, Glenford J. (1979): *The Art of Software Testing*, New York: John Wiley & Sons

National Research Council (2000): T*he Digital Dilemma. Intellectual Property in the Information Age*, Washington, D.C.: National Academy Press

National Research Council (1991): *Intellectual Property Issues in Software*, Washington, D.C.: National Academy Press

Office of Technology Assessment (1992): *Finding a Balance: Computer Software, Intellectual Property and the Challenge of Technological Change*, OTA-TCT-527, Washington, D.C.: Government Printing Office

Oppliger, Rolf (2003): *Sicherheit von Open Source Software*, in: Datenschutz und Datensicherheit (DuD), Jg. 27, 2003, Nr. 11, S. 669–675, http://www.dud.de/dud/documents/dud11-03.pdf

Parnas, David L., A. John van Schouwen und Shu Po Kwan (1990): *Evaluation of Safety-Critical Software*, Communications of the ACM, Vol. 33, Nr. 6, S. 636–648

Poulsen, Kevin (2004): *DARPA-funded Linux security hub withers*, in: Securityfocus News vom 30. Januar 2004,
online http://www.securityfocus.com/news/7947

Raymond, Eric S. (1999): *The Cathedral and the Bazaar*, S. 27–78, in: Eric S.Raymond: The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary, Sebastopol, CA: O'Reilly,
online http://www.openresources.com/documents/cathedral-bazaar/main.html

Robles, Gregorio, Hendrik Scheider, Ingo Tretkowski und Niels Weber (2001): *Who Is Doing It? A Research on Libre Software Developers*, Forschungsbericht, August 2001, Forschungsgebiet Informatik und Gesellschaft, Technische Universität Berlin,
online http://ig.cs.tu-berlin.de/s2001/ir2/ergebnisse/OSE-study.pdf

Ryan, Michael P. (1998): *Knowledge Diplomacy: Global Competition and the Politics of Intellectual Property*, Washington, D.C.: Brookings Institution Press

Sandhu, Ravi (2003): *Good-Enough Security*, in: IEEE Internet Computing, Vol. 7, No. 1 (January/February 2003), S. 66–68

Scacchi, Walt (2002): *Is Open Source Software Development Faster, Better, and Cheaper than Software Engineering?*, Paper presented at the 2nd ICSE Workshop on Open Source Software Engineering, May 2002, Orlando, FL

Schneier, Bruce (1996): *Angewandte Kryptographie: Protokolle, Algorithmen und Sourcecode in C*, 2. Aufl., Reading, MA: Addison Wesley

Schneier, Bruce (2000): *Secrets and Lies. Digital Security in a Networked World*, New York: John Wiley & Sons

Schneier, Bruce (2002): *Computer Security: It's the Economics, Stupid*, in: WEIS (2002)

Schulzki-Haddouti, Christiane (2001): *Das Ende der Schweigsamkeit: EU-Parlament verabschiedet Echelon-Untersuchungsbericht*, in: c't 19/2001, S. 44

Myung, Seung eun (2003): *Korea jettisons Windows for Linux*, in: silicom.com News, 1. Oktober 2003,
online http://www.silicon.com/news/500011-500001/1/6225.html?rolling=1

Stamelos, Ioannis, Lefteris Angelis, Apostolos Oikonomou und Georgios L. Bleris (2002): *Code quality analysis in open source software development*, in: Info Systems Journal, Vol. 12, Iss. 1, S. 43–60,
online http://www.blackwell-synergy.com/links/doi/10.1046/j.1365-2575.2002.00117.x/abs/

Stark, Jacqueline (2002): *Peer Reviews as a Quality Management Technique in Open-Source Software Development Projects*, in: Proceedings of Software Quality – ECSQ

2002: 7th International Conference, Helsinki, Finland, June 9–13, 2002, S. 340–350

Stobbs, Gregory A. (2000): *Software Patents*, 2. Aufl., Gaithersburg und New York: Aspen Law & Business

Tan, Bernard C. Y., H. Jeff Smith, Mark Keil und Ramiro Montalegre (2003): *Reporting Bad News About Software Projects: Impact of Organizational Climate and Information Asymmetry in an Individualistic and a Collectivistic Culture*, in: IEEE Transactions on Engineering Management, Vol. 50, No. 1, February 2003, S. 64–77

Thierer, Adam und Clyde Wayne Crews, Jr. (2003): *What's Yours is Mine: Open Access and the Rise of Infrastructure Socialism*, Washington, D.C.: Cato Institute

Tockey, Steve (1997): *A Missing Link in Software Engineering*, in: IEEE Software, Vol. 14, No. 6 (November/December 1997), S. 31–36

Tripp, Leonard L. (2002): *Benefits of Certification*, in: IEEE Computer, Vol. 35, No. 6, S. 31–33

Varian, Hal (2002): *System Reliability and Free Riding*, in: WEIS (2002).

Weinberg, Gerald M. (1971): *The Psychology of Computer Programming*, New York: Van Nostrand Reinhold

Wheeler, David A. (2003): *Why Open Soure Software / Free Software (OSS/FS)? Look at the Numbers!*, Abschnitt 6: Security,
online http://www.dwheeler.com/oss_fs_why.html

Wei-Ming, Tu (1993): *Way, Learning, and Politics: Essays on the Confucian Intellectual*, Albany, NY: State University of New York Press

WEIS (2002): *Proceedings of the 1st Workshop on Economics and Information Security*,
online http://www.sims.berkeley.edu/resources/affiliates/workshops/econsecurity/

WEIS (2003): *Proceedings of the 2nd Workshop on Economics and Information Security*,
online http://www.cpppe.umd.edu/rhsmith3/index.html

Zao, Luyin und Sebastian Elbaum (2003): *Quality Assurance und the open source development model*, in: The Journal of Systems and Software, Vol. 66, No. 1, S. 65–75

Zetter, Kim (2004): *Open-Source E-Voting Heads West*, in: Wired News, 21. Januar 2004,
online: http://www.wired.com/news/evote/0,2645,61968,00.html