universität wien

DIPLOMARBEIT

Titel der Diplomarbeit

„Free und Open Source Software – ein privat bereitgestelltes öffentliches Gut“

Verfasser

Matthias Nagl

angestrebter akademischer Titel

Magister der Sozial- und Wirtschaftswissenschaften

(Mag. rer. soc. oec.)

Wien, im März 2009

Studienkennzahl lt. Studienblatt:	A 140
Studiengebiet lt. Studienblatt:	Diplomstudium Volkswirtschaft
Betreuer:	Univ.-Prof. Mag. Dr. Gerhard Clemenz

Danksagung

Ich möchte mich zu Beginn dieser Arbeit bei all jenen Menschen bedanken, die zum Entstehen dieser Arbeit beigetragen haben. Das sind zum einen meine Eltern, die mich während des gesamten Studiums in jeder Art und Weise unterstützt haben. Zum anderen sind es jene Kolleginnen und Kollegen, die mich beim Schreiben der Diplomarbeit mit ihrem Fachwissen unterstützt haben und mir während des gesamten Studiums durch gemeinsames Lernen und Arbeiten das Weiterkommen erleichtert und zahlreiche mühselige Stunden verkürzt haben. Außerdem möchte ich Professor Gerhard Clemenz für die Betreuung dieser Arbeit danken. Im Zuge des von ihm geleiteten Universitätskurses „Network Economics“ bin ich erst auf dieses spannende Diplomarbeitsthema gestoßen. Und schließlich möchte ich mich noch bei meiner Familie und meinen Freundinnen und Freunden bedanken, deren jahrelange Unterstützung nicht hoch genug gewürdigt werden kann.

INHALTSVERZEICHNIS

1 Einleitung

Die technische Revolution in der EDV (elektronische Datenverarbeitung) und der Telekommunikation in den vergangenen 25 Jahren lässt praktisch keinen Lebensbereich unberührt. Abgesehen von den offensichtlichsten Veränderungen, wie der massiv gestiegenen Bedeutung von Computern oder der massenhaften Verbreitung der Mobiltelefonie, hat diese Entwicklung weniger sichtbare Folgen, die nicht minder bedeutend sind.
So erlangt an einer Schnittstelle zwischen EDV und Telekommunikation mit der Entwicklung von Free- und Open Source Software eine kooperative Produktionsweise aufgrund der gestiegenen Bedeutung von EDV signifikante Bedeutung. Durch die Entwicklung von Free- und Open Source Software wird praktisch ein öffentliches Gut von Privaten bereitgestellt. Das ist der ökonomischen Lehrmeinung folgend nur in Ausnahmefällen möglich.
Die vorliegende Arbeit will anhand eines Modells die Gründe für diesen konkreten Ausnahmefall untersuchen. Dabei sollen vorliegende Erklärungsmodelle mit neuen Denkansätzen verknüpft werden und mit theoretischen Ansätzen, die die private Bereitstellung von öffentlichen Gütern allgemein zu erklären versuchen, verglichen werden. Dabei ist es nicht das Ziel, eine singuläre Erklärung zu finden. Wie fast überall bedarf ein derart umfassendes Phänomen vielschichtiger Erklärungsmuster. Somit gibt diese Arbeit auch einen Überblick über bisher getätigte Erklärungsversuche und ergänzt diese.

Dazu wird zuerst der Untersuchungsgegenstand, Free- und Open Source Software, genau definiert und ein historischer Überblick über das Kulturgut Software im Allgemeinen und den Untersuchungsgegenstand im Speziellen gegeben. Im nächsten Kapitel wird Software als digitales Informationsgut definiert und ihre Eigenschaften werden beleuchtet. Anschließend wird der Begriff „Öffentliches Gut“ erklärt und argumentiert, warum es sich bei Free- und Open Source Software um ein öffentliches Gut handelt. Zudem werden ältere theoretische Ansätze, die private Bereitstellung öffentlicher Güter zu erklären, vorgestellt.
Das nächste Kapitel stellt den Hauptteil der Arbeit dar. Darin wird die Entwicklung von Free- und Open Source Software modelliert und es werden verschiedene Erklärungsversuche für die Entwicklung der Software vorgestellt und mit neuen Erklärungsversuchen verknüpft. Im letzten Kapitel werden die Motive von privaten

Softwareentwicklern, sich an der Entwicklung von F/OSS zu beteiligen, anhand eines Überblicks über verschiedene empirische Arbeiten vorgestellt.

2 Free/Open Source Software

Der Begriff der Open Source Software ist ein missverständlicher, der häufig falsch verwendet wird. Deshalb versuche ich zu Beginn der Arbeit eine Definition zu liefern, was unter Open Source Software zu verstehen ist und was nicht. Auf einer breiten Basis eingeführt wurde der Begriff Open Source Software für Software mit frei zugänglichem Quellcode 1998 durch die „Open Source Initiative“, dazu aber später mehr.

Open Source Software zeichnet sich dadurch aus, einen frei zugänglichen Quellcode (source code) zu haben. Daher auch der Name, der sich auf den Quellcode bezieht. Dieser Quellcode oder Quelltext bildet die Basis eines jeden Computerprogramms und ist quasi der Bauplan eines Programms in Programmiersprache. Für den Laien nicht verständlich kann ein Nutzer mit Informatik-Kenntnissen mit dem Quellcode das Programm exakt nachbauen oder seinen Bedürfnissen anpassen. Die Hardware[1] bildet aus dem Quellcode die Software, die auf dem Ausgabegerät erscheint und auch für Laien nützlich ist.
Theoretisch kann bei Open Source Software also jeder Benutzer das Programm nachbauen und beliebig abändern. Diese Software wird meistens unter Lizenzen veröffentlicht, die die Benutzer dazu verpflichten, ihre Änderungen ebenfalls wieder frei zugänglich zu machen. Die prominenteste derartige Lizenz ist die GNU[2] General Public License (GPL)[3]. Mittlerweile werden nicht nur Softwareprogramme unter solchen Lizenzen veröffentlicht, sondern auch Texte, Bilder oder Musik, die, unter der Bedingung die Veränderungen wieder frei zugänglich zu machen, ebenfalls frei verwendet und verändert werden dürfen. Die Gesellschaft „creative commons“[4] hat eine ganze Reihe von Lizenzen erstellt. Für diese Art von Informationsgütern, die auch als Kulturgüter bezeichnet werden könnten, würden sich zahlreiche weitere Forschungsfragen anbieten, die allerdings den Rahmen dieser Arbeit sprengen würden und denen deshalb nicht weiter nachgegangen wird.

---

1 In den meisten Fällen ein Computer, es kann sich aber auch um Server, Handys oder ähnliches handeln.
2 GNU ist ein Akronym und steht für „GNU is not Unix“, eine Referenz an das Betriebssystem Unix, zu dem mit GNU eine Alternative entwickelt werden sollte.
3 Diese Lizenz wird laufend überarbeitet. Die derzeit (Herbst 2008) aktuelle Version findet sich unter folgender Adresse: http://www.gnu.org/licenses/gpl-3.0.txt. Eine inoffizielle Übersetzung ins Deutsche gibt es hier: http://www.gnu.de/documents/gpl.de.html.
4 http://creativecommons.org/

Trotz der auf den ersten Blick großen Ähnlichkeit ist Open Source Software nicht mit Freeware zu verwechseln. Diese ist ebenso frei zugänglich und kostenfrei nutzbar, allerdings ist der Quellcode nicht öffentlich zugänglich, die Software kann also den eigenen Bedürfnissen nicht angepasst werden. Als Beispiel sei die beliebte Antiviren-Software „Avira AntiVir“ erwähnt, bei der es sich für Privatbenutzer um Freeware handelt. Eine andere beliebte Freeware ist das Programm „Google Earth“. Freeware, bei der dem Nutzer die Bezahlung für das Programm freigestellt bleibt, heißt Donationware. Große Ähnlichkeit besteht dagegen zwischen Open Source Software und Free Software, daher ist auch die Bezeichnung Free- and Open Source Software (F/OSS)[5] verbreitet. Technisch handelt es sich weitgehend um dasselbe, der Unterschied liegt eher in ideologischen Fragen. Anhänger der Open-Source-Bewegung gehen davon aus, dass OSS proprietärer Software überlegen ist und sich deshalb durchsetzt, Anhänger von Free Software legen großen Wert auf die freie Zugänglichkeit und Weiterentwickelbarkeit von Software nicht aus technologischen Gründen, sondern um Kontrolle über das Programm zu haben. Zur Unterscheidung der beiden Begriffe weiter unten mehr.

Eine Einschränkung der Freiheiten der meisten Lizenzen existiert bei halbfreier Software (semi-free software). Dabei hat der Nutzer die gleichen Rechte wie bei F/OSS, sprich der Quellcode ist ebenfalls öffentlich, die einzige Einschränkung ist das Verbot von kommerzieller Nutzung der Software. Das Verbot wurde eingeführt, nachdem in einigen Fällen kommerzielle Unternehmen F/OSS-Programme modifizierten und als proprietäre Software am Markt verkauften. Unter proprietärer Software versteht man klassische Software-Programme mit nicht-offenem Quellcode, deren Nutzungsrecht und meistens auch weiterführende Unterstützung man durch Zahlung erhält. Dabei gibt es noch die Untergruppe der Shareware, die grundsätzlich auch eine kommerzielle Basis hat. Eine zeitlich begrenzte Testversion oder eine nicht vollständige Teilversion des eigentlichen Programms ist aber frei zugänglich, der Quellcode bleibt dem Entwickler vorbehalten. Das Pack- und Entpackprogramm „WinRaR“ ist eines der beliebtesten Shareware-Programme.

[5] In dieser Arbeit wird für Free und Open Source Software die Abkürzung F/OSS verwendet.

2.1 Ein kurzer historischer Abriss

Um das Phänomen der Open Source Software zu verstehen, ist ein kurzer historischer Überblick über das Kulturgut Software unabdingbar, hat Open Source Software seine Wurzeln doch in der Frühzeit der Computergeschichte. Open Source Software ist keineswegs ein rezentes Phänomen, genau genommen waren die ersten Software-Programme allesamt F/OSS, proprietäre Software wurde erst später gängig. In der Frühzeit der Computer-Geschichte war der gegenseitige Austausch von Betriebssystemen, inklusive Quellcode, weit verbreitet. Dies geschah unter anderem vor dem Hintergrund der akademischen Überzeugung des freien Austauschs von Wissen, der viele der damaligen Akteure anhingen. Die ersten Betriebssysteme wurden an Universitäten wie der „University of California“ in Berkeley oder dem „Massachusetts Institute of Technology“ entwickelt. Zu dieser Zeit, in den 1960er und 70er Jahren, gab es kaum Versuche, die Software eigentumsrechtlich zu schützen, Software galt nicht als Ware. In der Frühzeit waren Hard- und Software keine getrennten Güter, sie wurden gemeinsam ausgeliefert. Das heißt, mit dem Erwerb des Computers erstand man eine passende Software automatisch dazu. IBM ging 1969 von dieser Praxis ab und entbündelte Soft- von Hardware. Hintergrund dürfte ein Kartellverfahren sein, das in den USA gegen IBM gestartet wurde und zu einer Zerschlagung des Konzerns hätte führen können.[6] Das war der Beginn von Software als eigenständiger Ware und für die Entstehung von reinen Softwarefirmen, die vorher nicht existierten und heute – erwähnt sei das Beispiel Microsoft – ein wichtiger Spieler am Markt sind.

Im Lauf der späten 70er und frühen 80er Jahre gab es erste Versuche, Eigentumsrechte für Software zu beanspruchen. Der Berühmteste der frühen Fälle ist jener des Unternehmens AT&T, das sein Betriebssystem UNIX[7], nachdem es auf den Markt kam, aufgrund einer Entscheidung der US-Monopolbehörde nur zum Selbstkostenpreis weitergeben durfte. Das stimulierte eine weitreichende Verbreitung der Software. Ab Ende der 70er Jahre versah AT&T das Programm mit zunehmend restriktiven Lizenzen. Damit ging die Weiterentwicklung des Programms, die vorher hauptsächlich die Nutzer vorantrieben, wieder ins Unternehmen zurück und die Lizenzierung führte zum vorläufigen Ende des freien Informationsflusses zwischen den Entwicklern und Nutzern und zu einer zunehmenden Kommerzialisierung von

[6] Stein (2006), S. 21

[7] UNIX ist Basis des heute populärsten OSS-Betriebssystems GNU/Linux. GNU/Linux ist ein weiterentwickelter Nachbau von UNIX.

Software. „Erst über die Verwendung von Lizenzen […], welche lediglich das Benutzen der Software erlaubten, jedoch ausschlossen diese zu untersuchen, zu verändern, weiterzugeben und in Verbindung mit der Zurückhaltung des Quellcodes […] konnte etwas wie ein Markt für Software entstehen.“[8]

Als Antwort darauf gründete der Informatiker Richard Stallman 1983 die Free Software Foundation (FSF), die das Ziel hatte, Software und deren Quellcode frei verfügbar zu machen und deren Mitglieder auch selbst derartige Software entwickelten. Primäres Ziel von Stallman war es gewesen, einen Ersatz für UNIX ohne einschränkende Lizenzbedingungen zu schaffen. Im Zuge dessen erstellte die FSF auch eine Lizenz für diese Software, die bis heute in Abwandlungen genutzte General Public License, die sichern sollte, dass die Software auch nach Veränderungen frei verfügbar und veränderbar bleibt. Das brachte dieser Art von Lizenz auch den Beinamen „copyleft“[9] ein, da sie die freie Verfügbarkeit von geistigem Eigentum sichern soll. Wobei das „Free“ in der Bezeichnung „Free Software” nicht für frei im Sinne von kostenfrei steht, sondern für die Freiheit des Entwicklers, zu entscheiden, was er damit macht.

> „Free software is a matter of liberty, not price. To understand the concept, you should think of free as in free speech, not as in free beer.”[10]

In den 80er Jahren blieb diese Bewegung freilich ein Minderheitenphänomen. Proprietäre Software dominierte den Softwaremarkt. Mit der steigenden Bedeutung des Internets in den 90er Jahren erlebte Open Source Software gewissermaßen eine Renaissance, wenngleich proprietäre Software den Markt weiterhin dominierte. Sowohl die Zahl der Entwickler als auch die Zahl der Benutzer von F/OSS stieg rapid an. Durch zahlreiche neue F/OSS-Projekte stieg auch die Bedeutung dieser Form von Software. Hauptgrund dafür war der durch die zunehmende Verbreitung des Internets stark gesunkene Preis für Kommunikation und Weiterverbreitung von Software. Dadurch beschleunigten sich sowohl Kommunikation als auch Distribution rapide, was die Bedeutung von F/OSS weiter steigerte.

---

8 Stein (2006), S. 37

9 „Copyleft“, da dem Nutzer das Recht zur Kopie überlassen wurde.

10 The Free Software Definition, http://www.gnu.org/philosophy/free-sw.html, 04.11.2008

2.2 Free Software vs. Open Source Software

Im Zuge dieser Entwicklung kam es innerhalb der Bewegung für quelloffene Software zu einer für die Benutzer unbedeutend erscheinenden Spaltung. Im Jahr 1998 bildete sich neben der FSF die Open Source Initiative (OSI), die den Begriff Open Source Software einführte, da sie davon ausging, dass der Begriff Free Software missverständlich sein könnte, was die kostenlose Weitergabe von Software betrifft. Durch die neue Bezeichnung erhoffte sich die OSI eine größere Verbreitung von F/OSS, was den empirischen Daten nach auch eintrat. Hintergrund der Spaltung ist ein Richtungsstreit zwischen verschiedenen Denkschulen, von denen die OSI vor allem für jene steht, „die vor allem die Vorteile des Entwicklungsmodells und der Technologie hervorhoben“[11], und die FSF für jene, „deren Priorität auf der Betonung der Freiheit des Individuums liegt und die einen gesellschaftsverändernden Anspruch haben.“[12]

Eine erste Entwicklung weg von den Ideen der FSF sieht Gerd Sebald in seiner Diskursanalyse „Offene Wissensökonomie“[13] bereits bei Linus Torvalds’ Anfängen der Entwicklung des F/OSS-Betriebssystems Linux in den Jahren 1991 und 1992. Torvalds ging es sowohl bei der Entwicklung des Systems als auch bei der Kommunikation darüber mit anderen Entwicklern stets um den Fortschritt der technischen Entwicklung. Er war zu Beginn der Entwicklungsarbeit auf der Suche nach einem Betriebssystem und hatte nie die Intention wie Stallman, die Welt zu verändern. Torvalds beschäftigte sich außerdem so wenig wie möglich mit Lizenzierungsfragen, es war ihm nur wichtig, die Kontrolle über das Projekt zu behalten.
Während die FSF um Stallman die Frühzeit der Softwareentwicklung mit dem freien Austausch von Betriebssystemen und Programmen als Idealzustand ansah und die Kommerzialisierung der Verteilung von Software per se ablehnte, sah Torvalds in einer verstärkten Ökonomisierung Chancen für Linux. Er war von der technischen Überlegenheit von Linux überzeugt, für ihn war der offene Quellcode weniger Zugeständnis an potenzielle Nutzer, als eine Offenheit gegenüber neuen Entwicklern. Entgegen dem gesellschaftsverändernden Anspruch der FSF ging es der OSI um die Verbreitung eines ihrer Ansicht nach überlegenen Software-Entwicklungsmodells. Bei

[11] Stein (2006), S. 45
[12] ebenda
[13] Sebald (2008), S. 59ff

der OSI ging es von Anfang an auch darum, Kapital zu lukrieren und für Unternehmen interessant zu sein, da nur so eine ernsthafte Verbreitung möglich wäre, so die Argumentation der OSI. Es wurde auch von einem eigenen Geschäftsmodell gesprochen, das F/OSS bieten würde. Über Dienstleistungen im Zusammenhang mit F/OSS ließe sich Geld verdienen. Wie schon erwähnt, gelang F/OSS das Überschreiten der Wahrnehmungsgrenze erst nach der Gründung der OSI, dieser Schritt kann also als geglückt bezeichnet werden.

3 Das Gut Software

Neben einer allgemeinen Definition, worum es sich bei F/OSS handelt, tut eine genauere ökonomische Definition, welches Gut Software ist, Not. Dazu reduziert man Software am besten auf das, was es im Grunde ist: eine Folge von Nullen und Einsen. Es handelt sich genau genommen also um die schlichteste Form der Information. Software wird den Informationsgütern zugerechnet. Diese haben ganz spezielle Eigenschaften, die später noch genauer beleuchtet werden. Zuerst folgt aber noch eine genauere Kategorisierung von Software innerhalb der Informationsgüter. Software ist Teil der digitalen Güter, einer Unterfamilie der Informationsgüter.

3.1 Digitale Güter

Digitale Güter zeichnen sich dadurch aus, eine Abfolge von Bits (also Nullen und Einsen), sogenannte Bitstrings zu sein, die Nutzen bringen. Dieser Nutzen kann vielfältiger Natur sein, Datensätze, Musik, Bilder oder ähnliches in digitaler Form zählen ebenfalls zu den digitalen Gütern. Derartige digitale Güter sind mittlerweile alltägliche Gebrauchsgüter. Danny Quah[14] unterscheidet unter den digitalen Gütern noch weiter zwischen fragilen und robusten digitalen Gütern. Letztere bleiben bei kleinen Änderungen am Bitstring unverändert und können auch nach geringen Schäden oder Veränderungen verwendet werden. Fragile digitale Güter verlieren bei der geringsten Veränderung ihre Funktionalität. Ein Beispiel dafür aus dem Bereich dieser Arbeit ist der sogenannte Maschinencode eines Programms. Der kommuniziert direkt mit der Recheneinheit eines Computers und ist für das Ausführen eines Programms elementar, dem Laien aber praktisch unbekannt. Sobald dieser Code verändert wird, ist das Programm nicht mehr funktionsfähig.
Typische Beispiele für robuste digitale Güter findet man in der Realität einer digitalisierten Gesellschaft praktisch überall, komprimierte Dateiformate wie MP3 (komprimierte Musikdateien) oder JPEG (komprimierte Bilddateien) sind klassische robuste digitale Güter. Das digitale Gut, in diesem Fall eine Audiodatei oder ein Bild, kann auch nach der Veränderung, der Komprimierung, wenn auch mit Abstrichen in

[14] Quah (2002), S. 6f.

der Qualität konsumiert werden. Digitale Güter haben eine Reihe von besonderen Eigenschaften, die sie deutlich von anderen Gütern abheben. Zum Ersten sind die Reproduktionskosten für digitale Güter sehr gering und bei Reproduktion entsteht kein Qualitätsverlust. Das heißt, dass diese Güter ohne großen Aufwand vervielfältigt werden können und jede Kopie dem Original entspricht. Daraus folgt, dass im Falle einer Weitergabe (oder eines Weiterverkaufs) des Gutes, der ursprüngliche Besitzer nicht auf das Gut verzichten muss. Digitale Güter sind also praktisch beliebig vermehrbar.
Sind die Produktionskosten ab der zweiten Einheit also nahe bei null, bilden die Kosten für die Produktion der ersten Einheit eines digitalen Gutes den Großteil der gesamten Produktionskosten. Diese Kosten werden gemeinhin als Entwicklungskosten bezeichnet. Diese Eigenschaften sind denen anderer Informationsgüter sehr ähnlich, Bücher beispielsweise zeichnen sich ebenfalls dadurch aus, geringe Reproduktionskosten und hohe Kosten für die erste Einheit zu haben. Sie gelten auch für die einfachste Form der Informationsgüter, Informationen oder besonders Ideen. Eine Idee ist ungleich schwieriger zu „produzieren“, als weiterzugeben.

Danny Quah schreibt digitalen Gütern in seinem Paper „Digital Goods and the New Economy“ fünf zentrale Eigenschaften zu, die das Wesen dieser Güter hinreichend erklären. Quah folgend, sind digitale Güter nicht-rivalisierend, unendlich erweiterbar, diskret, unräumlich und rekombinant. Nicht-Rivalität bedeutet, dass ein zusätzlicher Konsument des Gutes die Qualität des Gutes nicht beeinträchtigt. Bei digitalen Gütern, für die das Kriterium der Nicht-Ausschließbarkeit ebenfalls gilt, kann man also von öffentlichen Gütern sprechen. Digitale Güter sind per se eigentlich nicht ausschließbar und somit öffentliche Güter. Die Ausschließbarkeit kann aber eingefügt werden, entweder technisch durch Verschlüsselung oder legal durch Eigentumsrechte.
Die unendliche Erweiterbarkeit wurde in dieser Arbeit ebenfalls schon angesprochen. Digitale Güter können mit Grenzkosten von Null beliebig (also theoretisch unendlich) oft kopiert werden, die Qualität bleibt selbstverständlich konstant. Aber auch wenn man ein Softwareprogramm als digitales Gut als solches hernimmt, ist es theoretisch unendlich erweiterbar, wenngleich die Kosten dann nicht mehr Null betragen und pure unendliche Erweiterbarkeit somit nicht mehr gegeben ist. Durch Erweiterung mit zusätzlichen Features kann jede Software praktisch unendlich ausgebaut werden.

Die Diskretheit digitaler Güter ergibt sich auf den ersten Blick von selbst. Auf den zweiten Blick sind digitale Güter freilich nicht diskret, sie entfalten ihren Nutzen aber erst in diskretem Zustand. Macht eine Aussage wie „die Software ist zur Hälfte fertig“ Sinn, ist es logisch, dass ein Benutzer erst aus der vollständig fertigen, betriebsfähigen Software Nutzen zieht. Offensichtlicher wird die Diskretheit digitaler Güter mit einer anderen Interpretation dieser Eigenschaft, der Unteilbarkeit. Digitale Güter sind zwar beliebig vervielfältigbar, aber nicht teilbar. Dass fragile digitale Güter diskret sind, liegt auf der Hand.
Andererseits kann argumentiert werden, dass Software im speziellen nur bis zur Fertigstellung der ersten funktionsfähigen Einheit diskret ist. Zwar erfolgt die Weiterverbreitung und Reproduktion in ganzen, diskreten Einheiten. „Lebendige“ Software, die in ständiger Verwendung ist, wird laufend weiterentwickelt und verändert und ist somit nie „fertig“. Als praktische Beispiele dafür dienen Updates, die für jegliche Arten von Software angeboten werden, die Leistung verbessern und Fehlerquellen beseitigen sollen. Das gilt sowohl für F/OSS als auch für proprietäre Software. Der Entwicklungsprozess ist also nie wirklich abgeschlossen. Der Prozess der Softwareentwicklung kann als ein laufender, quasi unendlicher Prozess angesehen werden, der praktisch nie abgeschlossen wird. Nach dieser Interpretation kann Software als stetig angesehen werden. Andererseits treffen die Eigenschaften, die digitale Güter diskret machen, auch auf Software zu, Software kann also sowohl als diskret als auch stetig bezeichnet werden.

Eine weitere Eigenschaft digitaler Güter ist, dass sie unräumlich sind. Das bedeutet, dass ihre Verwendung nicht auf einen Ort beschränkt ist. Digitale Güter sind nicht greifbar und unsichtbar, höchstens die Datenträger auf denen sie gespeichert sind. Durch diese Eigenschaft ist auch die Weiterverbreitung über große Distanzen kein Problem. Die einzige Voraussetzung ist entsprechende Telekommunikation, um Daten zu übertragen.
Außerdem sind digitale Güter rekombinant. Sie können untereinander verbunden werden und daraus zusätzlichen Nutzen spenden. Diese Eigenschaft hat auch mit der oben angesprochenen ständigen Entwicklung von Software zu tun. Digitale Güter können immer mit anderen digitalen Gütern sinnvoll verbunden werden. In diesem Zusammenhang sei noch erwähnt, dass viele digitalen Güter ohne eine derartige Verbindung mit einem anderen digitalen Gut praktisch wertlos sind. Zum Beispiel ist eine Textverarbeitungssoftware ohne ein dazu passendes Betriebssystem praktisch

nutzlos. Ebenso verlieren digitale Musikstücke ihren Wert, wenn keine Software zum Abspielen der Musik zur Verfügung steht.

3.1.1 Die rechtliche Komponente digitaler Güter

Aus einer der bedeutendsten Eigenschaften digitaler Güter, den Grenzkosten von Null, also der ungehinderten Vervielfältigbarkeit, ergibt sich ein ernsthaftes Problem für die ökonomische Realität digitaler Güter, das sich auch in der Praxis zeigt. In einer kompetitiven Marktwirtschaft ist der Preis in einem Gleichgewicht gleich den Grenzkosten eines Gutes. Der Hersteller eines digitalen Gutes kann also unter Marktbedingungen nur einen Preis von Null verlangen. Da die Fixkosten (im Falle von Software die Entwicklungskosten) in der Regel natürlich größer sind, kann ein Hersteller seine Kosten nicht begleichen und digitale Güter können ökonomisch nicht effizient hergestellt werden. Das bedeutet weiter, dass es für Unternehmer keinen Anreiz gibt, derartige Güter zu entwickeln, da sie nicht gewinnbringend verkauft werden können. Innovation würde in einem derartigen Markt also nicht stattfinden.
Es gibt aber Möglichkeiten, diese Marktregeln außer Kraft zu setzen und Entwicklung zu stimulieren. Alle Rechte an geistigem Eigentum – Patente, Copyright, Markenschutz, etc. – zielen letztlich darauf ab, künstliche Monopole zu schaffen, um Entwicklung zu ermöglichen und zu belohnen. Der Inhaber des geistigen Eigentumsrechts an einem Gut kann einen Monopolpreis für sein Produkt festsetzen. Geistige Eigentumsrechte kommen fast ausschließlich im Zusammenhang mit Informationsgütern zur Verwendung. Denn auch wenn beispielsweise im Patentrecht bestimmte Güter patentiert werden, ist das, worauf der Schutz des Patents abzielt, die Idee dahinter und nicht das Produkt an sich. Freilich haben derartige Rechte den Nachteil, die Verbreitung der Idee oder des darauf aufbauenden Gutes zu behindern. Die genaue Ausgestaltung geistiger Eigentumsrechte, sodass sie insofern effizient sind, einerseits den Schöpfer der Idee zu schützen und andererseits die Verbreitung nicht zu behindern, ist also schwierig.

Dabei besteht bei Software noch ein weiteres Problem. Während Patente in der Regel für Ideen hinter greifbaren Gütern, wie Medikamente oder Maschinen, angewendet werden, hat Copyright seine Verwendung hauptsächlich für ungreifbare Güter (oft kreativ-geistige Erzeugnisse) wie Texte, Bilder oder Musikstücke. Wie Quah ausführt, ist Software beidem zuzurechnen: „ […] the difference between

patents and copyrights is that the former deals with machines, the latter with texts. Computer software and other digital goods erode the boundary between machines and texts."[15] Das erklärt zum Teil, warum zumindest die F/OSS-Welt ein anderes rechtliches System verwendet, das aber freilich auch auf den Ideen des geistigen Eigentumsrechts fußt.

3.2 Der Preis von Informationsgütern

Die zentrale Eigenschaft von Informationsgütern sind in der Produktion hohe Fixkosten und sehr niedrige Grenzkosten. Daraus folgt, dass die Preisfindung bei Informationsgütern nicht den herkömmlichen mikroökonomischen Regeln folgt, also der Preis aus den Grenzkosten hervorgeht. Carl Shapiro und Hal Varian geben in ihrem Buch „Information Rules" die Bewertung durch die Konsumenten als das zentrale Preisfindungselement an:

> „You must price your information goods according to consumer value, not according to your production cost. "[16]

Die Schwierigkeit dabei ist, den Wert eines Gutes für die Konsumenten idealerweise bereits vor dem Beginn der Entwicklung zu kennen. Das wäre deshalb wichtig, weil der Großteil der Fixkosten bei Informationsgütern versunkene Kosten sind. Das bedeutet, dass sie, sobald sie getätigt wurden, unwiederbringlich sind, also praktisch keinen dauerhaften Wert haben.
Eine Möglichkeit, das Problem der ungewissen Bewertungen der Konsumenten zu lösen, ist, das Informationsgut über den Preis zu differenzieren. Somit kann man zumindest für unterschiedliche Konsumentengruppen unterschiedliche Angebote bieten und unterschiedliche Preise verlangen. In der Praxis geschieht das meistens über verschiedene Versionen für ein bestimmtes Informationsgut. Beispielsweise wird bei Büchern meistens zuerst eine teurere Hardcover-Version veröffentlicht und etwas später eine billigere Taschenbuch-Version. Auch bei Software sind verschiedene Versionen weit verbreitet, dabei ergibt sich der Preis aber nicht nur aus der zeitlichen Verzögerung, sondern auch aus qualitativem Unterschied. Ein Beispiel

[15] Quah (2002), S. 25
[16] Shapiro/Varian (1999), S. 3

dafür wäre die weiter oben besprochene Softwareform Shareware.

Ein weiterer Punkt bei Informationsgütern, der auch für Software nicht unwesentlich ist, ist der Begriff der Meta-Information. Gewisse Informationsgüter bedürfen weiterer Informationen für den Konsumenten, um überhaupt konsumiert werden zu können. Eric Brousseau und Nicholas Curien[17] unterscheiden Informationsgüter zwischen puren Informationsgütern, für deren Konsum keine außergewöhnlichen Fähigkeiten notwendig sind, und informationsintensiven Gütern, deren Konsum zusätzliches Wissen benötigt. Letzteres müssen nicht notwendigerweise Informationsgüter sein, aber es gibt auch informationsintensive Informationsgüter.

Das hat über die Bewertung der Konsumenten freilich auch Auswirkungen auf die Preisfindung, bekommt ein Konsument zusätzliche Meta-Informationen, kann sich seine Bewertung für bestimmte Produkte ändern. Vereinfacht gesagt, bedeutet das, dass ein Produkt für eine Person einen höheren Nutzen hat, wenn sie davor lernt, damit umzugehen.

[17] In „Internet and Digital Economics“ (2007), S. 22ff

4 Öffentliche Güter

Öffentliche Güter zeichnen sich in erster Linie durch zwei Eigenschaften aus. Einerseits Nicht-Rivalität im Konsum, andererseits Nicht-Ausschließbarkeit. Diese beiden Kriterien definiert Paul M. Johnson wie folgt:

> Öffentliche Güter "... cannot practically be withheld from one individual consumer without withholding them from all and for which the marginal cost of an additional person consuming them, once they have been produced, is zero."[18]

Aufgrund der Nicht-Ausschließbarkeit muss das öffentliche Gut allen Individuen einer Gesellschaft in gleichen Maßen zugänglich sein, der Ausschluss von einzelnen Individuen vom Konsum dieses Gutes ist praktisch unmöglich. Geeignete Beispiele dafür sind die öffentlichen Güter der sauberen Umwelt oder der Landesverteidigung. Die Nicht-Rivalität verlangt, dass die Qualität des Guts von der Anzahl der Benutzer unabhängig ist. Außerdem muss das Gut von einer beliebigen Zahl an Benutzern gleichzeitig konsumiert werden können. Zusammenfassend könnte man die beiden Kriterien so umschreiben wie bei Varian: Das Gut muss „allen betroffenen Konsumentinnen im selben Ausmaß zur Verfügung gestellt werden"[19].
Während bei einigen Autoren die Erfüllung beider Kriterien Grundbedingung für die Definition als öffentliches Gut ist, kann allgemein je nach Erfüllung der Kriterien über die Qualität des öffentlichen Gutes unterschieden werden. Sind beide Kriterien erfüllt, spricht man von einem reinen öffentlichen Gut. Ist die Bedingung der Nicht-Rivalität im Konsum nicht erfüllt, spricht man von einem unreinen öffentlichen Gut. Eine Straße zur Hauptverkehrszeit ist ein Beispiel für ein unreines öffentliches Gut. Wenn die Straße überfüllt ist und es zu Staus kommt, sinkt die Qualität des Gutes durch zusätzliche Benutzer. Ein Beispiel aus dem Forschungsbereich dieser Arbeit ist der Fall, wenn sehr viele Benutzer gleichzeitig ein Softwareprogramm aus dem Internet herunterladen. Die Downloadgeschwindigkeit kann sich stark verringern. Das beeinträchtigt zwar die Qualität des Gutes an sich nicht, bedeutet aber niedrigere Qualität zumindest in der Beschaffung des Gutes.

---

[18] http://www.auburn.edu/~johnspm/gloss/public_goods, 5.3.2008
Definition by Dr. Paul M. Johnson, University of Auburn

[19] Varian (2001), S.604

Das ökonomische Problem bei öffentlichen Gütern ist, dass eine ökonomisch herkömmliche Preisfindung über Angebot und Nachfrage kaum möglich ist. Die beiden Kriterien sorgen zwar dafür, dass es ausreichend Interessenten für das Gut gibt, deren Zahlungsbereitschaft aber relativ gering ist, da das Gut, wird es bereitgestellt, auch von theoretisch allen anderen Konsumenten konsumiert werden kann. Letzteres Phänomen – ein Gut zu konsumieren, ohne dafür bezahlt zu haben – wird als Trittbrettfahren bezeichnet.
Darum gilt, dass außer in Ausnahmefällen öffentliche Güter nicht ökonomisch effizient bereitgestellt werden können. Durch die Neigung zum Trittbrettfahren ist die Kostendeckung unwahrscheinlich. Somit geht die klassische Literatur davon aus, dass öffentliche Güter vom Staat bereitgestellt werden müssen, was sich abgesehen von Ausnahmefällen auch in der Praxis zeigt. Häufig genannte Beispiele für öffentliche Güter sind Straßenlaternen, Landesverteidigung und saubere Umwelt, allesamt Güter, die nicht privat bereitgestellt werden.

4.1 Open Source Software als öffentliches Gut

Nimmt man die beiden Hauptkriterien, Nicht-Ausschließbarkeit und Nicht-Rivalität im Konsum als Maßstab, kann F/OSS als öffentliches Gut definiert werden. Die Nicht-Ausschließbarkeit ergibt sich erst aus der Eigenschaft von F/OSS, frei zugänglich zu sein, und gilt für proprietäre Software nicht. Durch entsprechende Kosten (den Preis), Passwörter oder ähnliche digitale Sperren, kann proprietäre Software potenziellen Konsumenten vorenthalten werden. Somit gilt das Kriterium der Nicht-Ausschließbarkeit nur für F/OSS. Nicht-Rivalität im Konsum gilt im Grunde für jegliche Software. Unabhängig von der Anzahl der Benutzer bleibt die Qualität erhalten. Die Entwicklungskosten, die bei Software meistens die Fixkosten ausmachen, bleiben konstant, egal ob man die Software für einen Benutzer entwickelt oder für unendlich viele. Ein reines öffentliches Gut ist lediglich F/OSS, auf dieser Art von Software liegt auch der Fokus dieser Arbeit.
In der Praxis braucht es aber freilich einige Bedingungen, die erfüllt sein müssen, um F/OSS als öffentliches Gut definieren zu können. So muss die notwendige Infrastruktur für die jeweilige Software vorhanden sein. Neben einem adäquaten

Internetzugang, um leichten Zugang zur Software zu haben, muss ein Benutzer eine zur Software kompatible Hardware haben, um einen Nutzen aus dem Konsum ziehen zu können. Zudem benötigt man zum Konsum von F/OSS die zum Konsum von informationsintensiven Informationsgütern nötige Meta-Information. In einfachen Worten heißt das, dass ein Benutzer von F/OSS mit der entsprechenden Hardware umgehen können muss und auch wissen muss, wie die Software zu bedienen ist, um aus dem Konsum Nutzen zu haben. Das ist speziell bei F/OSS zum Teil ein Ausschlusskriterium, da es sich bei F/OSS oftmals um recht komplexe Software handelt. Sind diese Bedingungen erfüllt, ist die gängige Definition eines öffentlichen Gutes zutreffend. Da die Bedingungen nicht direkt mit der Bereitstellung in Zusammenhang stehen, haben sie auf die Definition des eigentlichen Gutes als öffentlich auch keinen direkten Einfluss.

4.2 Die private Bereitstellung von öffentlichen Gütern

Die Frage nach der privaten Bereitstellung öffentlicher Güter ist in der ökonomischen Literatur nicht erst mit F/OSS aufgekommen. Christopher Bliss und Barry Nalebuff beschäftigten sich in ihrem Beitrag „Dragon-Slaying and Ballroom-Dancing“ im Jahr 1984 mit der Frage privater Bereitstellung öffentlicher Güter. Dieser Beitrag im „Journal of Public Economics“ bildet auch die Basis für mehrere Artikel zum Thema F/OSS als öffentliches Gut und kann als wegweisend auf dem Gebiet privater Bereitstellung öffentlicher Güter bezeichnet werden. Bliss und Nalebuff ziehen dafür abstrakte Güter als Beispiele heran. Sie sehen ihr Modell anwendbar auf Situationen wie das Spenden für eine Bibliothek, das Öffnen eines Fensters in einem heißen Raum oder das Retten eines Schwimmers in Seenot. Andere Beispiele, wo öffentliche Güter von privater Hand bereitgestellt werden, sind: die Finanzierung von politischen Parteien (das gilt vor allem für den US-amerikanischen Raum, aus dem der Großteil der Literatur zum Thema private Bereitstellung öffentlicher Güter kommt. Mittlerweile sind private Spenden für politische Parteien aber auch hierzulande ein wichtiger Faktor in der Finanzierung von Parteien und Interessengruppen) sowie Auslandshilfe, „Internationale öffentliche Güter“ oder Friedenssicherung (auch wenn in diesen Bereichen hauptsächlich Staaten tätig sind, ist es theoretisch dem Handeln privater Akteure recht ähnlich. Es gibt ebenfalls mehrere Akteure (Staaten), die von der Handlung eines Einzelnen oder Mehrerer einen Nutzen haben).

Es geht also um Situationen, wo das Handeln eines Einzelnen oder einiger Weniger erforderlich ist, um Vielen einen Nutzen zu bringen. Diese Idee ließe sich auch auf F/OSS übertragen, wenngleich dieses Beispiel weniger dramatisch ist, als das der Rettung eines Schwimmers in Seenot.

Im Modell von Bliss und Nalebuff haben die n Akteure jeweils Kosten von c für die Versorgung mit dem öffentlichen Gut. Die Wartezeit, bis ein Akteur agiert, hängt monoton von den Kosten sowie von der Anzahl der beteiligten Akteure ab. Je länger ein Akteur wartet, desto höher ist sein Informationsgrad. Wenn zu einem Zeitpunkt, wo seine Kosten bei $c*$ liegen, noch niemand das öffentliche Gut bereitgestellt hat, weiß er, dass die Kosten aller anderen Akteure über $c*$ liegen. Jeder Akteur wählt seine optimale Wartezeit der monotonen Funktion $T(n+1,c)$ folgend.

Bliss und Nalebuff lösen das Optimierungsproblem durch Ableitung und kommen zu folgenden Ergebnissen: Eine größere Population kommt Akteuren mit höheren Kosten eher entgegen, da diese größeres Augenmerk darauf legen, dass sie die Versorgung mit dem öffentlichen Gut später leisten müssen und die Wahrscheinlichkeit, dass sie gar nicht agieren müssen, höher bewerten. Ihre Chance, als Trittbrettfahrer agieren zu können, erhöht sich also. Durch einen zusätzlichen Akteur sinken die erwarteten Kosten der anderen Akteure, also hat jeder eine größere Chance, als Trittbrettfahrer agieren zu können.

Es ist allerdings kein Gesetz, dass das öffentliche Gut bei einer größeren Population früher bereitgestellt wird. Die Autoren beweisen aber, dass das Gut, wenn die Populationsgröße ins Unendliche wächst, zu Kosten $\hat{c}$ bereitgestellt wird. $\hat{c}$ sind dabei die minimalen Kosten mit positiver Dichte, also die kleinstmöglichen Kosten. Wenn $\hat{c} = 0$, ist auch das Trittbrettfahrerproblem gelöst, da ein Akteur keine Kosten und also auch keinen Anreiz, Trittbrett zu fahren, hat.

Wie später zu sehen sein wird, hängt auch die Bereitstellung des öffentlichen Guts F/OSS von den Kosten und der Anzahl der einzelnen Akteure ab. Wenngleich die Idee des Modells von Bliss und Nalebuff also durchaus zur Erklärung des Phänomens F/OSS beitragen kann, ist das Modell selbst für die Frage nach der Bereitstellung von F/OSS mit Vorsicht zu genießen. Die beiden Autoren machen Annahmen, die das Modell für F/OSS eigentlich unanwendbar machen. Im Modell ist der Wert des öffentlichen Guts für alle Akteure konstant gleich eins. Das trifft auf F/OSS nicht zu, ein derart komplexes Gut wird praktisch von jedem Akteur unterschiedlich bewertet. Auch wenn es verschiedene Akteursgruppen mit

unterschiedlichen Programmierfähigkeiten geben mag, ist die konkrete Bewertung eines Projekts auch innerhalb dieser Gruppen unterschiedlich. Diese Annahme gilt aufgrund der Komplexität des Guts also nicht für F/OSS.

Theodore Bergstrom, Lawrence Blume und Hal Varian haben sich ungefähr zur gleichen Zeit in ihrem Beitrag „On The Private Provision of Public Goods“ ebenfalls mit diesem Thema befasst. In ihrem Modell gehen sie von einem öffentlichen und einem privaten Gut aus. Die Individuen können ihr Vermögen zwischen dem Konsum des privaten Gutes und einer Spende für das öffentliche Gut aufteilen. Im ersten Teil ihres Modells befassen sie sich mit der Frage, wie sich die Bereitschaft für Spenden für das öffentliche Gut ändert, wenn Einkommen umverteilt wird. Das zentrale Resultat von Bergstrom, Blume und Varian ist, dass sich die Versorgung mit dem öffentlichen Gut nicht ändert, wenn die Menge der spendenden Individuen gleich bleibt.
Für diese Arbeit interessant sind aber eher die weiterführenden Resultate, im speziellen die Frage nach den Konsequenzen eines Einstiegs der öffentlichen Hand in die Finanzierung eines öffentlichen Gutes. Die Autoren zeigen, dass es, wenn sich die öffentliche Hand an der Bereitstellung eines vormals privat bereitgestellten öffentlichen Gutes beteiligt, zu einer teilweisen Verdrängung der privaten Spenden, einem sogenannten partiellen Crowding-Out[20] kommt. Dabei wird der Anteil des Staates an der Finanzierung über eine Steuer finanziert.
Wenn die Steuer, die von einem bestimmten Individuum eingehoben wird, dessen freiwillige Spende nicht übersteigt, verringern sich die aggregierten privaten Spenden um genau den Betrag, den der Staat beisteuert. Das Individuum wird seinen Spendenumfang nämlich genau um den Betrag der Steuer reduzieren. Das gesamte Aufkommen für das öffentliche Gut bleibt also gleich. Es erhöht sich, wenn die öffentliche Hand einen Teil der Steuern von Individuen einhebt, die bisher nichts zum öffentlichen Gut beigetragen haben. Die freiwillige Spende eines Einzelnen kann natürlich trotzdem sinken, es kann trotzdem zu Crowding-Out kommen. Falls die Steuer höher ist, als die freiwillige Spende einzelner Individuen, wird sich der Gesamtbetrag für das öffentliche Gut ebenso erhöhen. Inwieweit es also zu einem Crowding-Out kommt und wie groß dessen Betrag ist, hängt von der Höhe der Spenden sowie der Steuer ab. Später werden wir sehen, dass es auch im Fall von F/OSS zu Crowding-Out-Effekten kommen kann.

[20] Ein einfaches Crowding-Out wäre eine vollständige Verdrängung.

5 Ein F/OSS-Modell

Im Folgenden werde ich ein Modell präsentieren, dass eine Kombination von Modellen von James Bessen[21], Justin P. Johnson[22], Richard Schmidtke[23] und S. E. Holtermann[24] darstellt. Dieser Ansatz wurde gewählt, da die einzelnen Modelle jeweils nur einen Aspekt behandeln, diese Arbeit aber einen vielschichtigeren Erklärungsversuch unternehmen will. Das kombinierte Modell behandelt die Entwicklung einer speziellen Software für ein bestimmtes Unternehmen. Dabei werden verschiedene Varianten der Entwicklung von Software modelliert, darunter F/OSS, und gezeigt, dass F/OSS zu einer Pareto-Verbesserung der Gesamtsituation führt. In diesem Zusammenhang wird auch auf die Trittbrettfahrerproblematik und mit neuen Denkansätzen auf ihre Besonderheiten im Fall von F/OSS eingegangen. Es wird ein Vergleich angestellt, inwieweit F/OSS im Vergleich mit kommerzieller Software konkurrenzfähig ist und F/OSS wird als Komplementärgut zu einem privaten Gut betrachtet. Außerdem wird überprüft, inwieweit F/OSS als positive Externalität zu betrachten ist, ein Punkt, der in der behandelten Literatur unterrepräsentiert ist. Weiters wird ein Engagement der öffentlichen Hand bei der Entwicklung von F/OSS beleuchtet.

5.1 Getrennte Softwareentwicklung

Eine Firma benötigt eine bestimmte Software. Die Firma hat nach dem Modell von Bessen zwei Möglichkeiten, die benötigte Software zu erlangen. Entweder der Betrieb wendet sich an eine Software-Entwicklungsfirma, lagert also die Entwicklung der Software aus (e). Oder entwickelt beispielsweise durch eine hauseigene IT-Abteilung die Software selbst (E). Die Entscheidung wird in der Praxis natürlich unter anderem von den Fähigkeiten der hauseigenen IT im Vergleich mit einer Entwicklungsfirma abhängen. Bessen nimmt der besseren Verständlichkeit des Modells zuliebe aber an, dass die Benutzerfirma über die gleichen Entwicklungsfähigkeiten wie die Entwicklerfirma verfügt.

[21] Aus dem Paper „Open Source Software: Free Provision of Complex Public Goods“
[22] Aus dem Paper „Open Source Software: Private Provision of a Public Good“
[23] Aus dem Paper „Private Provision of a Complementary Public Good“
[24] Aus dem Paper „Externalities and Public Goods“

Entscheidet sich die Firma für die Variante mit ausgelagerter Entwicklung, muss ein Vertrag mit der Entwicklerfirma über die genaue Ausgestaltung des Produkts geschlossen werden. Das gestaltet sich aufgrund der Komplexität des Gutes Software schwierig. So geht es bei der Vertragsgestaltung eher um eine Idee, wie die Software aussehen soll, als um das genaue Produkt, wie Bessen beschreibt:

> „That is, typically the developer gets a general idea of how the customer wants the program to work, then makes educated guesses about how the interactions should be coded, and allows the customer to review and modify these decisions."[25]

Bessen modelliert die Komplexität folgendermaßen: Eine Software hat M mögliche Features, m davon sind für die Benutzerfirma von Interesse. Jedes Feature kann mit jedem anderen Feature auf zwei mögliche Arten interagieren, es gibt also 2^m verschiedene Arten die m interessanten Features zu verwenden. Das heißt, die Verwendungsmöglichkeiten steigen mit der Anzahl der interessanten Features exponentiell an.

Entsteht auf einem der zwei möglichen Wege eine funktionierende Software, die den Vorstellungen des Unternehmens entspricht, zieht die Benutzerfirma einen Nutzen (V) daraus. Die Wahrscheinlichkeit, dass die Entwicklung gelingt, wird beschrieben durch die Funktion $p = p(e + E)$ [26]. Entwickelt die Softwarefirma erfolgreich, bietet sie das Produkt gegen eine Lizenzgebühr y an. Sie maximiert dann ihren Nutzen: $p(e + E)y - e$, das ist der erwartete Umsatz minus den Kosten der Entwicklung. Die Benutzerfirma maximiert ebenfalls ihren Nutzen: $p(e + E)(V - y) - E$, der erwartete Nutzen minus der Lizenzgebühr und der eigenen Entwicklungskosten.

Wenn die Benutzerfirma selbst entwickelt und auf die Dienste der Softwarefirma verzichtet, ist die Maximierungsfunktion: $p(E)V - E$. Gelingt die Entwicklung der Software, ist das ein sozial optimales Resultat. Wenn die interne Entwicklung der Software scheitert, wird die Benutzerfirma mit einer Entwicklungsfirma in Verhandlungen treten und im Verhandlungsprozess werden die Firmen

---

[25] Bessen (2003), S. 11

[26] p ist steigend und konkav, $p(0) = 0; p'(0) = \infty; p < 1$

übereinkommen, den Nutzen zu teilen. Jede Firma erhält im Erfolgsfall also $\left(\frac{V}{2}\right)$. Die Maximierungsfunktionen im Gleichgewicht sehen folglich so aus: $\left[p(e+E)\frac{V}{2} - e \right]$ für die Entwicklerfirma und $\left[p(e+E)\frac{V}{2} - E \right]$ für die Benutzerfirma. Es ist offensichtlich, dass dieses Ergebnis sozial nicht optimal ist. Bei gleichen Entwicklungsfähigkeiten (wie oben angenommen) wird die Benutzerfirma die Software selbst entwickeln. Durch die Verhandlungen zwischen den Firmen kommt es also zu einem sozial nicht optimalen Resultat.

Bessen verweist zur Erklärung dieses Umstands auf Aghion & Tirole[27], die in ihrem Paper zeigen, dass die Benutzerfirma erhöhte Verhandlungsmacht hat, da sie der einzige mögliche Abnehmer der Software ist. Die Entwicklerfirma würde also von einer höheren Anzahl an möglichen Benutzern auf zwei Arten profitieren. Einerseits durch erhöhte Verhandlungsmacht, andererseits durch die Möglichkeit, das fertige Produkt auch anderen Abnehmern zu verkaufen, also höhere Erlöse durch die Lizenzgebühren zu generieren. Das bringt uns zur zweiten, weiterentwickelten Variante:

5.2 Proprietäre Software[28]

Das Modell wird um die Möglichkeit mehrerer potenzieller Softwarekunden erweitert. n gibt die erwartete Anzahl der Firmen, die eine bestimmte Software benötigen, an. Hier muss man aber die Komplexität des Gutes Software beachten. Eine Entwicklungsfirma kann nicht wissen, wie viele potenzielle Abnehmer es für eine spezifische Software geben würde.

> „If a developer cannot know the “special widget” customer A needs, how can the developer know that customer B and C also need the same special widget? […] each firm will have highly idiosyncratic needs.”[29]

[27] Aghion/Tirole (1994)

[28] Dieser Terminus ist in der Diskussion über die verschiedenen Softwareformen gebräuchlich. Er beschreibt Software, die von ihren Entwicklern herkömmlich eigentumsrechtlich (nicht mit GPL oder ähnlichen im Laufe der Geschichte der Softwareentwicklung entstandenen Lizenzen) geschützt wird und deren Quellcode nicht frei ist.

[29] Bessen (2003) , S. 13

Es wird aber trotzdem immer Features geben, die für die meisten Firmen eines bestimmten Marktes nützlich sind. Die n erwarteten Firmen bei Bessen beziehen sich auf die m am meisten nachgefragten Features einer bestimmten Software.
Die Entwicklungsfirma investiert also einen bestimmten Betrag (e) in die Entwicklung der Software und beschließt die Anzahl der geplanten Features m. Die potenziellen Benutzerfirmen entscheiden, ob das Produkt ihren Bedürfnissen entspricht und investieren einen bestimmten Betrag (E) in die interne Entwicklung einer eigenen Software. Sollte dieses Projekt scheitern, haben sie immer noch die Möglichkeit, die Software der Entwicklungsfirma zum Preis (w) zu kaufen.

Das Softwareprodukt entspricht den Wünschen der Benutzerfirmen unterschiedlich gut, sie haben daher unterschiedliche Nutzenniveaus V_i [30] durch die Software. Die Entwicklerfirma[31] hat für eine Software mit m Features Kosten von $e \cdot (c_o + c2^m)$. Dabei sind c_o die Kosten, das Programm ursprünglich zu programmieren, und c die Kosten, das Programm den unterschiedlichen Bedürfnissen der Benutzer anzupassen und das Funktionieren trotzdem sicherzustellen. Das Unternehmen setzt einen Preis (w) für das Softwareprodukt fest. Dann ist $G(w)$ [32] der Anteil an potenziellen Kunden, die bereit sind, das Produkt zu kaufen, für die also $V_i > w$ gilt, und deren interne Entwicklung mit Kosten von $E \cdot (c_0 + c)$ gescheitert ist[33]. Daraus ergibt sich der optimale Preis für die Entwicklerfirma: $[wG(w)]$
Wenn $V_i < w$ oder wenn die angebotene Software nicht den Vorstellungen der Firma entspricht, ist ihre einzige Möglichkeit, interne Entwicklung mit dem Netto-Nutzen $[p(E)V_i - E \cdot (c_o + c)]$ zu betreiben. Die Entwicklerfirma maximiert ihren erwarteten Gewinn:

$$[p(e)n\hat{w}G(\hat{w}) - e \cdot (c_0 + c2^m)]$$

In Worten ist das die Wahrscheinlichkeit, dass die Entwicklung erfolgreich ist, mal

---

[30] V_i ist nach einer Verteilungsfunktion $F(\bullet)$ über $0 < V_i \leq \overline{V}$ verteilt und mit m nicht korreliert.

[31] Es wird angenommen, dass aufgrund des Bertrand-Wettbewerbs nur jeweils eine Firma eine bestimmte Spezifikation einer Software entwickelt. Ein Eintritt in den Markt würde sich für einen Rivalen nicht rechnen.

[32] G ist abhängig von F und davon, ob die Benutzer interne Entwicklung betreiben.

[33] Die beiden Bedingungen werden in der Formel $G(w) = (1 - F(w))(1 - p(\hat{E}(w)))$ ausgedrückt.

dem Preis mal dem Anteil der Käufer mal der Anzahl der potenziellen Käufer minus den Entwicklungs- und Erhaltungskosten.

Gegenüber dem vorigen Fall kommt es zu einer sozialen Verbesserung. Wenn eine Benutzerfirma das Produkt kauft, ist ihr Nutzen größer, als wenn sie es selbst entwickelt. Lediglich im Fall zu hoher Kosten $(V_i < \hat{w})$ oder hoher Ansprüche an das Produkt $(m* < m)$ wird die Benutzerfirma in die eigene Entwicklung investieren. Dieses Resultat führt uns schon in Richtung eines zentralen Ergebnisses der Arbeit. Man kann das Resultat so interpretieren, dass Institutionen oder Personen mit beschränkten finanziellen Mitteln und mit komplexen Anforderungen an Software eher dazu tendieren würden, selbst Software zu entwickeln oder an einem Software-Projekt mitzuarbeiten[34]. Diese Interpretation deckt sich mit Resultaten empirischer Studien zum Entwicklerkreis von F/OSS.

5.3 Software-Baukasten

Das Basisprogramm der kommerziellen Software des vorigen Falles kann von der Softwarefirma wieder genutzt werden. Wenn die Software funktioniert und erfolgreich etabliert ist, kann sie mit Kosten von c_{ext} erweitert werden. Der optimale Einsatz, den ein Benutzer in eine solche Erweiterung der Software investiert ist: $[p(E)V_i - Ec_{ext}]$.

Solange $c_{ext} < c_0 + c$ [35], ist der Nutzen für die Benutzerfirma mit dieser Variante größer, als im vorigen Fall, wo die Firma die Software komplett alleine entwickeln muss. Für die Benutzerfirma wäre diese Möglichkeit also eine Pareto-Verbesserung der Situation. Einzige Voraussetzung dafür ist, dass das Unternehmen von der Entwicklerfirma das Basisprogramm der kommerziellen Software bereitgestellt bekommt.

Dafür gibt es, Bessen folgend, zwei Möglichkeiten. Die beiden Unternehmen können wie im ersten Fall des Modells verhandeln. Sie werden den Nutzen in den

---

[34] Lediglich, dass Benutzer mit geringer Wertschätzung für die Software (niedriges V_i ; dazu weiter unten mehr) selbst in Entwicklung investieren, ist unlogisch. Bessen argumentiert, dass diese Benutzer in einem realistischeren Modell durch eine Randlösung nicht entwickeln würden.

[35] Diese Annahme scheint plausibel, da es billiger ist auf ein vorhandenes Basisprodukt aufzubauen als ein Produkt neu zu entwerfen

Verhandlungen wie schon im ersten Fall teilen, jedes Unternehmen wird also $\left(\frac{V}{2}\right)$ erhalten. Die Entwicklerfirma wird also jenes (e) in die Erweiterung der Software investieren, das $p(e)\frac{V_i}{2} - e \cdot c_{ext}$ maximiert. Wenn nun $2c_{ext} < c_0 + c$, dann ist der Nutzen für die Benutzerin größer, wenn sie die Software von Grund auf selbst entwickelt. Ob die Verhandlungsvariante profitabel ist, hängt von den Kosten der verschiedenen Entwicklungsvarianten ab, sowie von den Kosten, die sowohl die Entwicklungsfirma als auch die Benutzerfirma für die Softwareentwicklung haben.
Die Softwarefirma kann neben dem fertigen, kommerziellen Produkt aber auch das Basisprogramm mit einem API-Zugang[36], einer Art Entwickler-Baukasten, verkaufen. Der Preis für diesen Baukasten sei w_{API}. Entscheidet sich die Benutzerfirma für diese Variante, ist ihr Gewinn

$$[p[E]V_i - E \cdot c_{ext}] - w_{API} .$$

Abhängig vom Preis für den Baukasten und die Kosten für die interne Entwicklung kann diese Möglichkeit gewissen Benutzerfirmen eine Pareto-Verbesserung bringen. Die Zielgruppe für diese Variante sind Firmen, deren Bedürfnisse komplexer als die Möglichkeiten mit fertiger, kommerzieller Software sind, denen die Kosten für die interne Entwicklung der kompletten Software aber zu hoch sind. Der Ansatz mit der Erweiterung eines Basisprogramms funktioniert, weil die Basiskosten der Software, c_o, durch die Wiederverwendung des Basiscodes quasi eingespart werden. Der soziale Gewinn ist die Differenz zwischen der von der Basissoftware ausgehenden Entwicklung im Gegensatz zur vollständigen Entwicklung einer Software. Diese zusätzliche Möglichkeit kann aber wie die vorigen Ansätze trotzdem nicht alle potenziellen Benutzer zufrieden stellen.

[36] API steht für „application programming interface“, was mit Programmierschnittstelle übersetzt werden kann. Mit einem API-Zugang wie in diesem Modell verwendet, kann man auf Teile des Programmiercodes eines an sich fertigen Programms zugreifen und das Programm den eigenen Bedürfnissen entsprechend verändern. Unter dem Label Windows API bietet Microsoft Entwicklern und Entwicklerfirmen beispielsweise die Möglichkeit, ihre Programme für das Windows-Betriebssystem zu adaptieren.

5.4 Entwicklung von F/OSS

Die Baukasten-Variante hat gewisse Ähnlichkeiten zu einem F/OSS-Projekt. Ein solches, in dem die Basisarbeit bereits erfolgreich geleistet wurde, kann als Software mit API-Zugang betrachtet werden. Mit dem Unterschied zum vorigen Fall, dass keine zusätzlichen Kosten entstehen $(w_{API} = 0)$. Angenommen, die Software des F/OSS-Projekts hat die gleichen $m*$ Features, wie die fertige kommerzielle Software (oder auch mehr Features, beispielsweise wie eine Software, die mit API-Zugang entstanden ist). Dann steht die Software allen potenziellen Benutzerfirmen mit komplexeren Anforderungen zur Verfügung, allerdings ohne zusätzliche Kosten. Das ist eine weitere Pareto-Verbesserung der Situation.
Ein beliebiges F/OSS-Projekt, das $\tilde{m}$ Features hat, kann zu Kosten von c_{FOSS} um ein Feature erweitert werden. Wenn nun eine potenzielle Benutzerfirma, der diese Software nutzen bringt, zu den $\tilde{m}$ Features ein zusätzliches braucht, kann sie dieses durch eine Investition von $E \cdot c_{FOSS} = k$ in die interne Entwicklung selbst entwickeln. Ihr Gewinn ist dann:

$$\left[p(E)V_i - E \cdot c_{FOSS}\right]$$

Da bei Bessen der Blick auf die Beweggründe eines einzelnen Unternehmens fehlt, wird das Modell im Folgenden um eine auf Johnsons Modell basierende Entscheidungsregel erweitert. Die Firma wird die Erweiterung entwickeln, wenn ihr erwarteter Gewinn den erwarteten Nutzen übersteigt: $V_i - k > p(E)V_i$, oder wenn anders ausgedrückt: $\frac{V_i}{k} > \frac{1}{1-p(E)}$. In Worten ausgedrückt bedeutet das, die Firma entwickelt, wenn ihr Nutzen-Kosten-Verhältnis größer ist als der Kehrwert der Wahrscheinlichkeit[37], dass die Programmentwicklung scheitert. Für den gegenteiligen Fall gilt, dass die Benutzerfirma die Erweiterung nicht entwickeln würde. Der kritische Wert, bei dem $\frac{V_i}{k} = \frac{1}{1-p(E)}$, die Firma also indifferent ist, wird als $\hat{q}$ bezeichnet. Die

[37] Der Kehrwert einer Wahrscheinlichkeit $\left(x = \frac{1}{p}\right)$ sagt aus, dass das fragliche Ereignis bei jeder x-ten Durchführung eintritt. In diesem Fall bedeutet das, bei jedem wievielten Entwicklungsversuch die Entwicklung scheitert.

Verteilung der Werte des Nutzen-Kosten-Verhältnisses ist dann $F(q) = \Pr.\left\{\frac{V}{k} < q\right\}$.

Nun wird angenommen, dass es verschiedene Firmentypen gibt. Solche, die die Software sehr hoch bewerten (V_H), und solche, denen die Software weniger Wert ist (V_L)[38]. Sowie solche, die für die Entwicklung der Software hohe Kosten (k_H) haben, und solche, die Software relativ kostengünstig (k_L) entwickeln können. $q_H = \frac{V_H}{k_L}$[39] ist dann der obere Grenzwert des Nutzen-Kosten-Verhältnisses und kann als obere Grenze der Bereitschaft zur Unterstützung eines F/OSS-Projekts angesehen werden. Ob ein Unternehmen die Entwicklung von F/OSS unterstützt hängt also – wenig überraschend – von der Relation des Nutzens zu den Kosten ab. Die kumulierte Unterstützung wird also umso größer sein, je nützlicher ein Programm für eine möglichst große Zahl von Firmen ist und je geringer der nötige Aufwand für eine Beteiligung ist. Letzteres hängt natürlich auch von den Entwicklungsfähigkeiten innerhalb der Firma ab. Daraus folgt, dass F/OSS-Projekte – vor allem für komplexe Anwendungen – am ehesten von Firmen mit hoher Programmierkompetenz unterstützt werden. Auf einzelne Entwickler umgemünzt bedeutet das, dass Entwickler mit fortgeschrittener Entwicklungsfähigkeit die Software eher entwickeln, da anzunehmen ist, dass die Kosten für die Entwicklung indirekt proportional zur Entwicklungsfähigkeit sind.

5.4.1 Das Trittbrettfahrer-Problem

Es ist jedoch bei F/OSS nicht unüblich, dass Unternehmen oder Individuen aus der Software Nutzen ziehen, die zur Entwicklung der Software nichts beitragen. Sehr viele F/OSS-Programme sind problemlos über das Internet zu beziehen, ohne in irgendeiner Form zur Entwicklung beizutragen. In der klassischen Theorie über öffentliche Güter spricht man dabei vom Phänomen des Trittbrettfahrens. Wie Johnson gezeigt hat, ist dieses Problem im Bereich von F/OSS aber weniger dramatisch als bei anderen öffentlichen Gütern.
Zur kurzen ökonomischen Erklärung der Problematik des Trittbrettfahrens: Jedes öffentliche Gut bietet Gelegenheit zum Trittbrettfahren. Es wird zumeist ein Gut bereitgestellt, von dessen Konsum in der jeweiligen Gesellschaft niemand

[38] Auf die Verteilung der V_i `s umgemünzt, liegt ein V_H also nahe bei $\overline{V}$ und V_L relativ knapp über 0.
[39] Dieser Wert kann per definitionem nicht ins Unendliche gehen.

ausgeschlossen werden kann. Bereitsteller ist oft – direkt oder indirekt – die Gesellschaft als Ganze. Wenn ein Individuum weiß, dass das Gut ohnehin bereitgestellt wird, kommt es in Versuchung, das Gut auch zu konsumieren, wenn es nichts dazu beigetragen hat.
Tritt dieser Fall ein, spricht die ökonomische Theorie vom Trittbrettfahrer-Phänomen. Die klassischen Beispiele öffentlicher Güter eignen sich wieder zur Veranschaulichung: Es kann in unserer Gesellschaft niemand vom Konsum des Lichts der Straßenlaternen oder der Landesverteidigung ausgeschlossen werden, egal ob der- oder diejenige durch Steuerzahlungen zur Bereitstellung der Güter beiträgt. Das gilt auch für F/OSS-Projekte. Egal, ob jemand an der Entwicklung einer bestimmten Software mitarbeitet, oder ihre Entstehung durch Spenden unterstützt oder eben nicht. Das Individuum oder die Organisation kann die Software trotzdem herunterladen und verwenden.

Bessen behandelt die Frage der Trittbrettfahrer nur am Rande. Johnsons Paper ist für diese Frage wesentlich hilfreicher. Er zeigt, dass die Entwicklungswahrscheinlichkeit für F/OS-Software mit einer größeren Entwicklerbasis (oder Teilnehmerzahl) zwar sinken kann, dass dieser Rückgang aber nicht sehr groß ist: p_n sei die Wahrscheinlichkeit, dass, aus Perspektive einer beliebigen Benutzerfirma einer F/OSS, zumindest ein anderer Akteur die gefragte Erweiterung der Software entwickelt. Aus dem Modell wissen wir, dass der kritische Wert, an dem die Firma zwischen „entwickeln“ und „nicht entwickeln“ indifferent ist, $\hat{q} = \frac{1}{1-p_n} = (1-p_n)^{-1}$ ist. Die Verteilung der Werte des Nutzen-Kosten-Verhältnisses ist bekannterweise $F(q) = \Pr\left\{\frac{V}{k} < q\right\}$. Daraus folgt, dass die Wahrscheinlichkeit, dass außer einem bestimmten Benutzer niemand anderer entwickelt, $F(\hat{q})^{n-1} = 1-p_n$ ist. Dann ist $p_n = 1-F(\hat{q})^{n-1} = 1-F\left(\frac{1}{1-p_n}\right)^{n-1}$. Da die Funktion $F(x)^{n-1}$ für jeden beliebigen Wert x in n abnimmt, nimmt p_n zu. Logischerweise nimmt dann auch $q_n\left(\frac{1}{1-p_n}\right)$ zu.
Wie sich die gesamte Entwicklungswahrscheinlichkeit mit steigender Teilnehmerzahl verhält, ist zwar weiter fraglich, die Wahrscheinlichkeit, dass aus der Perspektive

eines beliebigen Nutzers zumindest ein anderer Entwickler die Erweiterung entwickelt, steigt mit der Anzahl der teilnehmenden Individuen. Auch der kritische Wert des Nutzen-Kosten-Verhältnisses steigt im Gleichgewicht. Daraus leitet Johnson ab, dass die gesamte Entwicklungswahrscheinlichkeit nicht sehr viel abnehmen kann, wenn die Zahl der Entwickler eine gewisse Größe erreicht hat:

> "Conceptually, each agent contributes only a small amount to the probabilistic chance of development when the number of developers is not too small. Since the preceding lemma shows that, from the standpoint of a given agent, the chance of at least one of the other agents developing is increasing, it stands to reason that the overall development probability cannot go down by very much when the size of the developer base is not too small."[40]

Johnson berechnet dazu auch einen Höchstwert, über den der Rückgang der Entwicklungswahrscheinlichkeit bei einem zusätzlichen Entwickler nicht hinausgeht. Dieser Wert ist $\frac{1}{[(n-1)e]}$ und geht schnell gegen Null. Bei großen F/OSS-Projekten ist ein Rückgang der Wahrscheinlichkeit also unwahrscheinlich.

Dabei ist zu beachten, dass hier die Anzahl der potenziellen Entwickler als Basis genommen wird. Es gilt also nur als Trittbrettfahrer, wer auch tatsächlich die Möglichkeit dazu hätte, zur Entwicklung der Software beizutragen. Denn in der Praxis der F/OSS-Welt gibt es gewissermaßen eine große Zahl an Trittbrettfahrern. Jeder Benutzer, der die Software verwendet, aber nichts zu ihrer Erweiterung oder Entstehung beiträgt, ist im Grunde ein Trittbrettfahrer. Bei diesen Individuen handelt es sich aber oft um solche, die von der Entwicklung praktisch ausgeschlossen sind, da sie die Fähigkeit zu programmieren nicht besitzen. Es sind also nicht Trittbrettfahrer im ursprünglichen Sinne, sie könnten, auch wenn sie wollten, oft gar nicht zur Entwicklung beitragen (oft wissen sie nicht einmal, dass es sich bei der Software um F/OSS handelt, sie also theoretisch beitragen könnten). Nichtsdestotrotz gibt es oftmals Benutzer, die die Fähigkeiten zur Teilnahme an der Entwicklung hätten, dies aber trotzdem nicht tun. Durch die sehr große Zahl an potenziellen Benutzern die das Internet bietet, fallen die Trittbrettfahrer aber nicht ins Gewicht. Dies zur praktischen Erklärungen der oben angeführten Theorie.

[40] Johnson (2002), S. 645

5.4.1.1 Die Rolle der Bug-Reporter:

Dennoch tragen auch Benutzer, die selbst keine Programmierfähigkeiten haben, zur Entwicklung bei. Dieser Umstand wurde in der bisherigen Literatur zum Thema fast gänzlich ignoriert. Das hat vermutlich auch damit zu tun, dass diese Benutzer erst ab einem gewissen Entwicklungsstand beitragen. Bei der Entwicklung vom leeren Blatt weg sind tatsächlich nur Programmierer engagiert. Sobald eine Software aber in der Beta-Phase[41] der Entwicklung ist, werden auch die „normalen" Benutzer für die Entwicklung der Software wichtig. Sie erfüllen die Rolle der sogenannten „Bug-Reporter", jener Benutzer, die Fehler der Software melden. Das geschieht zumeist via E-Mail oder über Internetforen, wird teilweise aber auch über automatisierte Programme abgewickelt. Freilich handelt es sich dabei nicht ausschließlich um weniger Software-affine Benutzer. Auch potenzielle Entwickler oder Programmierer, die an der Entwicklung der Software mitarbeiten, erfüllen diese Rolle. Im Jänner 2009 sorgte Microsoft für Aufsehen, als es eine Beta-Version seines Betriebssystems „Windows 7" zum Download bereitstellte. Durch diesen Schritt hoffte der Konzern zahlreiche Fehlerquellen im Gegensatz zu früheren Versionen des Betriebssystems bereits vor der Veröffentlichung zu entdecken.

Obwohl die Bug-Reporter an der Entwicklung der Software nicht direkt mitarbeiten, sind sie auch keine Trittbrettfahrer. Ihr Beitrag zur Bereitstellung des öffentlichen Gutes Software ist letztendlich nur aufgrund der Komplexität von Software möglich. Denn die Aufgabe der Bug-Reporter ist auch nach der Veröffentlichung einer Software nicht erledigt. Die Komplexität von Software bringt es mit sich, dass eigentlich nicht von „fertiger Software" gesprochen werden kann. Es tauchen immer wieder neue Fehler auf, sei es durch Kombination mit anderen Programmen, sei es in der ursprünglichen Software selbst. Der Prozess der laufenden Entwicklung von Software wird durch die „Berichte" der Bug-Reporter maßgeblich geprägt. Ein weiteres Charakteristikum, der Bug-Reporter ist, dass sie nicht aktiv auf Fehlersuche sind. Die Fehler tauchen während der Verwendung der Software auf, stellen ein Problem für die Reporter dar und werden deshalb – mit der Hoffnung auf baldige Lösung – den Entwicklern gemeldet. Das gilt im Übrigen nicht nur für F/OSS. Auch

[41] Terminus für die fortgeschrittene Entwicklung einer Software. Während die Alpha-Phase noch ein sehr frühes Entwicklungsstadium bezeichnet, wird die Software ab der Beta-Phase bereits einem beschränkten Anwenderkreis zum Testen zur Verfügung gestellt.

Anbieter proprietärer Software versuchen, sich den Erfahrungsschatz der Benutzer in der Verwendung der Programme zu Nutzen zu machen.
Somit bedürfte auch die Definition des Trittbrettfahrens im Zusammenhang mit F/OSS einer Überarbeitung. Denn die klare Unterscheidung zwischen Trittbrettfahrer und Beiträger ist in der Form nicht passend. Es gibt einerseits potenzielle Beiträger (Entwickler), die tatsächlich Trittbrettfahren, also die Software benutzen, ohne etwas beigetragen zu haben, obwohl sie die Gelegenheit dazu hätten[42]. Und andererseits Benutzer, die zur Entstehung der Software aus praktischen Gründen nichts beitragen, aber dennoch die Weiterentwicklung indirekt unterstützen. So kann es also zu einer Situation kommen, in der ein Trittbrettfahrer das Gut konsumiert (die Software nutzt) und durch diesen Konsum zur Weiterentwicklung des Gutes beiträgt, das Trittbrettfahren global gesehen also einen positiven Einfluss auf das Gut hat. Die Komplexität des Gutes Software bringt es also mit sich, dass die herkömmliche Definition von Trittbrettfahren nicht mehr greift.
Somit mag Johnsons Ansatz in sich zwar richtig sein. Der Begriff „free-riding“, wie Johnson ihn verwendet, trifft die Frage aber nicht exakt. Bei der Bereitstellung von F/OSS ist die Abgrenzung zwischen Trittbrettfahrern und Beiträgern nicht klar zu ziehen.

5.4.2 Modularität

Neben dem Vorteil der großen Anzahl an potenziellen Entwicklern und Beiträgern, ist die Möglichkeit, eine Software einem Puzzle gleich aus verschiedenen, lose zusammenhängenden, kleinen Teilen zusammenzubauen, ein Grund für die erfolgreiche Etablierung von F/OSS neben proprietärer Software. Diese Möglichkeit besteht selbstverständlich bei proprietärer Software auch, aber erst durch den großen Pool potenzieller Entwickler wird es ein Vorteil für F/OSS-Projekte. Johnson hat diesen Ansatz der Modularität in seinem Modell näher beleuchtet, ich habe seine Erkenntnisse für das vorliegende Modell adaptiert.
Wir gehen wieder von dem zuletzt verwendeten Modell aus. Es gibt zwei Möglichkeiten, eine Software zu entwickeln, auf modulare und auf nonmodulare

[42] Wobei hier genau genommen eine weitere Unterteilung möglich ist. Auch einem ausgebildeten Programmierer kann es in der Praxis schwer möglich sein, an gewissen Projekten mitzuwirken. Dies kann an aufwändigen Einarbeitungsphasen, unterschiedlichen Programmiersprachen oder ähnlichem liegen.

Weise. Für die Entwicklung einer bestimmten Software sind (a) Aufgaben[43] zu erledigen. Jede Benutzerfirma[44], die wieder selbst entwickeln kann, hat einen bestimmten Nutzen sowie bestimmte Kosten für jede der Aufgaben. Die modulare Variante ist ein Nachbau der im bisherigen Modell angewendeten Software. Im modularen Modell erhalten alle Benutzer und Entwickler ihren Nutzen aus der jeweiligen Aufgabe, sobald sie von einem einzelnen Entwickler erledigt wurde. Im nonmodularen Modell erfolgt die Auszahlung erst, wenn zumindest ein Entwickler alle a Aufgaben erledigt hat. Daraus folgt, dass ein Entwickler quasi vor Arbeitsbeginn entscheiden muss, ob er alle Aufgaben erledigt, oder keine. In diesem Fall folgt der Entwickler der Entscheidungsregel $\left(q_i^a = \frac{\sum_{j=1}^{a} v_i^j}{\sum_{j=1}^{a} c_i^j} \right)$[45], wobei $\left(v_i^j\right)$ der Nutzen aus einer bestimmten Aufgabe ist, und $\left(c_i^j\right)$ die Kosten sind, die er für die Erledigung der jeweiligen Aufgabe zu leisten hat. Wenn diese Relation groß genug ist, entscheidet sich der Entwickler dazu, die Software zu entwickeln.

p_{mod}^n sei die Wahrscheinlichkeit, dass aus Sicht eines beliebigen Benutzers ein anderer (also einer von $n-1$ Benutzern) eine Aufgabe zur modularen Entwicklung einer Software erledigt. $p_{n\text{mod}}^{n,a}$ beschreibt die Wahrscheinlichkeit, dass die Software auf nonmodularem Weg erfolgreich entwickelt wird. Durch eine einfache mathematische Erweiterung kann die Entscheidungsregel der Entwickler auch wie folgt ausgedrückt werden: $\left(q_i^a = \frac{\frac{1}{a}\sum_{j=1}^{a} v_i^j}{\frac{1}{a}\sum_{j=1}^{a} c_i^j} \right)$. Wenn a sehr groß wird, die zu entwickelnde Software also verhältnismäßig komplex ist, dann ist dem Gesetz der großen Zahlen[46] folgend die Wahrscheinlichkeit, dass $\left(q_i^a = \frac{E(v)}{E(c)} \right)$ ist, sehr groß.

---

[43] Nicht zu verwechseln mit m, den Features. Ein Feature kann aus mehr als einer Aufgabe bestehen, und eine Aufgabe kann mehr umfassen als die Entwicklung eines Features.

[44] Es kann sich freilich auch um Privatbenutzer handeln, das ist nicht zuletzt von der Beschaffenheit der Software abhängig

[45] Diese Formel ist eine Weiterentwicklung der Nutzen-Kosten-Relation aus dem bisherigen Modell. Deshalb auch die Notation mit q.

[46] Das Gesetz der großen Zahlen besagt, dass sich bei einer ausreichend großen Menge an Versuchen die relative Häufigkeit eines Ereignisses der Wahrscheinlichkeit annähert. In diesem Fall, dass die

Auch die Verteilungsfunktion $F_a\left(q_i^a\right)$ konvergiert zu einer Funktion, die nahe bei $\frac{E(v)}{E(c)}$ liegt. Deshalb muss, wenn $\left(q_i^a = \frac{E(v)}{E(c)}\right)$ gilt, im nonmodularen Fall jede Lösung für $\left(p = 1 - F_a\left(\frac{1}{1-p}\right)^{n-1}\right)$ [47] sehr nahe an $1 - \frac{E(c)}{E(v)}$ liegen. Daraus folgt, dass $p_{n\mathrm{mod}}^{n,a}$ bei fixem (n)[48] zu $1 - \frac{E(c)}{E(v)}$ konvergiert, wenn a sehr groß wird, also je komplexer die zu entwickelnde Software ist.

Für den Fall, dass $\left(1 - p > F_a\left(\frac{1}{1-p}\right)^{n-1}\right)$[49], dann gilt auch $p_{\mathrm{mod}}^{n} > p$ sowie vice versa.

Wenn, wie oben angenommen $p = 1 - \frac{E(c)}{E(v)}$, dann gibt es einen kritischen Grenzwert der Entwicklerzahl eines Projekts, $N*$, für den gilt: $n = 1 + \frac{\log\left(\frac{E(c)}{E(v)}\right)}{\log\left(F\frac{E(v)}{E(c)}\right)} = N*$. Wenn nun $n > N*$, die Zahl der Entwickler an dem Projekt also den Grenzwert übersteigt, dann ist $p_{\mathrm{mod}}^{n} > p_{nmod}^{n,a}$, die Wahrscheinlichkeit, dass die Software auf modularem Weg entwickelt wird, also größer als auf nonmodularem Weg. Es gibt auch einen Grenzwert A' der zu erledigenden Aufgaben, für den gilt, wenn $n > N*$ und $a > A'$, dass $p_{\mathrm{mod}}^{n} > p_{nmod}^{n,a}$ ist, die Entwicklungswahrscheinlichkeit auf modularem Weg also größer ist. Im Fall, dass $n < N*$ und $a > A'$, dann gilt $p_{n\mathrm{mod}}^{n,a} > p_{mod}^{n}$. Die Wahrscheinlichkeit, dass das gesamte Projekt auf nonmodularem Weg entwickelt wird, ist in diesem Fall (relativ wenige Entwickler, relativ komplexe Anforderungen) also größer als die Wahrscheinlichkeit, dass eine Aufgabe im modularen Fall erledigt wird.

Welche der beiden Entwicklungswahrscheinlichkeiten größer ist, hängt also von den Grenzwerten der Entwicklerbasis, $N*$, sowie der Anzahl der zu erledigenden

---

Entscheidungsregel die Division der beiden Erwartungswerte ist.

47 Also jede Lösung für die Entwicklungswahrscheinlichkeit im nonmodularen Fall.

48 Sprich bei unveränderbarer Entwicklerzahl.

49 Dass die Wahrscheinlichkeit, dass die Entwicklung der Software scheitert, mit der Mitarbeit des jeweiligen Benutzers größer ist als ohne ihn.

Aufgaben, A', ab. Wenn die Zahl der Entwickler groß ist, ist der modulare Weg erfolgversprechender, sind nur wenige Entwickler verfügbar, ist der nonmodulare Weg erfolgversprechender. Zudem kann Nonmodularität den Anreiz zum Trittbrettfahren verringern:

> "Thus nonmodularity will sometimes temper the free-riding present in the open source development system."[50]

Da ein Programm bei nonmodularer Entwicklung erst einsatzfähig ist, wenn alle Aufgaben erledigt sind, ist der Anreiz für Entwickler, auch die unbeliebten Aufgaben, die bei modularer Entwicklung eventuell dem Trittbrettfahren zum Opfer fallen, zu erledigen, größer, argumentiert Johnson.
Diese Ergebnisse sind mit den weiter oben erzielten konsistent. Zusammenfassend scheint es naheliegend, dass komplexere Anwendungen, die nur für einen kleinen Benutzerkreis interessant sind, von wenigen, spezialisierten Entwicklern in F/OSS-Projekten entwickelt werden. Je einfacher die Anwendungen werden, desto breiter wird die Entwicklerbasis und desto eher wird das Programm auf modularem Weg entwickelt.

5.4.3 F/OSS als Komplementärgut

Die Eigenschaft von Software, dass sie praktisch nur als Komplementärgut[51] zu Hardware produziert wird, kann die Bereitstellung von F/OSS zum Teil erklären. Während Bessen und Johnson diese Erklärungsmöglichkeit gänzlich außer Acht lassen, ist sie bei Schmidtke der zentrale Ansatz. Dieser wird in der Folge in das vorliegende Modell integriert. Dazu wird angenommen, dass es zu F/OS-Software ein Komplementärgut gibt. Mehrere Firmen können in die Entwicklung der Software investieren. Sei es durch Investition in ein F/OSS-Projekt, oder sei es durch Mitarbeit firmeninterner Entwickler an einem Projekt.
Jedes an der Software interessierte Unternehmen kann also, wie im ursprünglichen Modell, wenn in die selbständige Entwicklung einer Software investiert wird, einen bestimmten Betrag e_i in die Entwicklung einer Software stecken. Die Qualität der

[50] Johnson (2002), S. 660
[51] Komplementärgüter zeichnen sich dadurch aus, dass sie gemeinsam konsumiert werden. Die Nachfragekurve hat also einen ähnlichen Verlauf wie die Nachfragekurve des Komplementärguts. Die Nachfrage nach dem Gut hängt auch mit dem Preis eines dazugehörigen Komplementärguts zusammen.

Software ergibt sich dann aus der gesamten Investitionssumme: $Q = \sum_{i=1}^{n} e_i$ [52]. Als Beispiel für ein Komplementärgut bietet sich im Fall von F/OSS Hardware (Computerserver, Drucker, Handys, etc.) an, deren Nachfrage nicht zuletzt vom Vorhandensein geeigneter Software abhängt. Eine Hardware produzierende Firma hat also ein Interesse daran, dass es passende, qualitativ hochwertige Software zum eigenen Produkt gibt. Es kann also angenommen werden, dass eine solche Firma in die Entwicklung von F/OSS investieren würde. So haben sich beispielsweise mehrere Unternehmen aus dem Umfeld der Mobilfunk- und Softwareindustrie zur „Open Handset Alliance“ zusammengeschlossen, um ein Betriebssystem für Handys auf F/OSS-Basis zu entwickeln.

$x_i \in [0; \bar{x}_i]$ sei die Produktion des privaten Komplementärguts zu Software von Firma i. Da es sich um Komplemente handelt, steigt die Zahlungsbereitschaft der Kunden für die Hardware, wenn die Qualität der dazugehörigen Software steigt. Die Nachfragefunktion nach dem privaten Gut, in unserem Beispiel also nach der Hardware, ist: $d = d(x_i, e_i)$, wobei $\frac{\partial d}{\partial x_i} < 0, \forall i \in \{1,...,n\}$ und $\frac{\partial d}{\partial e_i} > 0, \forall i \in \{1,...,n\}$. Die Nachfrage nach dem privaten Gut hängt also von der Investition in die Entwicklung der Software ab.

Daraus ergeben sich die Umsatzfunktion $R_i(x_i, e_i) = x_i d(x_i, e_i)$ und die Gewinnfunktion von i: $\pi_i = x_i d(x_i, e_i) - c(x_i) - c(e_i)$[53], wobei $c(x_i)$ die Kostenfunktion des privaten Guts[54] ist und $c(e_i)$ die Kosten für das Investment in das öffentliche Gut sind.

Bei steigenden Grenzkosten des öffentlichen Gutes existiert ein Gleichgewicht, in dem die Firmen $x*$ des privaten Gutes produzieren und $e*$ in das öffentliche Gut investieren. Dazu müssen die Grenzkosten des privaten Gutes, also für Hardware, gleich ihrem Grenzprofit sein. Außerdem hat das Investment in das öffentliche Gut lediglich indirekten Effekt auf den Profit, was aber auch intuitiv logisch erscheint. Sind die Grenzkosten des öffentlichen Gutes konstant, gibt es unendlich viele Gleichgewichte.

Unternehmen, die ein Komplementärgut zu Software herstellen, haben durch den

---

[52] Diese Annahme mag etwas verkürzt sein. Die Qualität hängt in der Regel nicht direkt von der Investitionssumme ab. Der besseren Verständlichkeit des Modells zuliebe wird sie aber beibehalten.

[53] Diese Funktion ist kontinuierlich und wie die Umsatzfunktion strikt konkav.

[54] Es wird angenommen, dass diese Funktion für alle Firmen gleich und konvex ist. Die Kostenfunktion des öffentlichen Gutes ist ebenfalls konvex.

indirekten Einfluss der Qualität von Software auf die Nachfrage nach ihrem privaten Gut einen Anreiz, in die Entwicklung des öffentlichen Gutes F/OSS zu investieren.

5.4.4 F/OSS als positive Externalität

Ein Aspekt von F/OSS, der in der gängigen Literatur fast gänzlich fehlt – alle bisher in dieser Arbeit behandelten Ansätze gehen darauf in keiner Weise ein –, ist jener der Auswirkungen von F/OSS auf die Produktion von anderen Gütern. Mit anderen Worten, F/OSS kann als positive Externalität[55] in der Produktionsfunktion anderer Güter betrachtet werden. Ich habe ein Modell zum Thema „Externalities and Public Goods“ von S. E. Holtermann in diese Arbeit eingefügt.
Externalitäten sind in der ökonomischen Theorie Outputs eines Akteurs, die Auswirkungen auf den Konsum oder die Produktion eines anderen Akteurs haben und sowohl positiv als auch negativ sein können. S.E. Holtermann definiert Externalitäten folgendermaßen:

> „... an externality exists whenever an output of one economic agent appears as an input in the consumption or production vector of another economic agent without any compensation being paid by either party...“[56]

Ich argumentiere im Folgenden kurz, unter welchen Umständen F/OSS als positive Externalität zu sehen ist. Wie bereits besprochen, muss F/OSS nicht notwendigerweise kostenfrei weitergegeben werden. Der zentrale Punkt in der Definition von F/OSS ist nicht der Preis von Null. Es gibt F/OSS, die nicht kostenfrei weitergegeben wird, wie es proprietäre Software gibt, die zu Kosten von Null weitergegeben wird. Somit ist für die Eigenschaft als positive Externalität nicht entscheidend, ob es sich um F/OSS handelt oder nicht, sondern, ob der Preis Null beträgt oder nicht. Wenn, wie in den meisten Fällen, der Preis von F/OSS Null ist, kann die Software als positive Externalität betrachtet werden. Sobald in der Produktion eines beliebigen Gutes kostenfreie F/OSS eingesetzt wurde, handelt es sich bei der jeweiligen Software nicht mehr um einen Produktionsfaktor, sondern um eine positive Externalität. Das gilt natürlich auch für proprietäre Software, sofern es sich um kostenfreie Software handelt.

---

[55] Externalitäten werden auch als externe Effekte bezeichnet.
[56] Holtermann (1972), S. 79

$(x_1,..,x_n)$ seien beliebige private Güter, die von beliebigen Akteuren produziert werden. $(x_{F/OSS})$ sei das öffentliche Gut, in diesem Fall eine F/OS-Software, die, wie oben gezeigt wurde, produziert wurde. Die Eigenschaft von F/OSS als öffentliches Gut ist dabei nicht unerheblich, Holtermann nimmt an, dass viele externe Güter[57] die Eigenschaften von öffentlichen Gütern haben. Das Modell von Holtermann ist insbesondere deshalb auch für F/OSS geeignet, da F/OSS die Bedingung erfüllt, dass die Gesamtversorgung mit dem öffentlichen Gut gleich groß ist wie der individuelle Konsum jedes Akteurs, der das Gut konsumiert. Es gibt s Akteure, die eine Nutzenfunktion von $u^i(x_1^i...x_n^i, x_{F/OSS})$ haben. Der soziale Nutzen ergibt sich aus dem Nutzen der Akteure: $U = U(u^1..u^s)$. In einer Transformationsfunktion werden die totalen Outputs und Inputs der privaten und öffentlichen Güter in Beziehung gesetzt: $F(x_1...x_{F/OSS}) = 0$. Diese Funktion ist in dem Modell deshalb von Bedeutung, da bei Externalitäten davon ausgegangen wird, dass ein Output eines Akteurs ein Input in der Produktionsfunktion eines anderen Akteurs ist.

Die für das öffentliche Gut zentrale Bedingung für maximalen sozialen Nutzen im Gleichgewicht lautet wie folgt: $\sum_i^s \frac{u_{F/OSS}^i}{u_r^i} = \frac{F_{F/OSS}}{F_r}$. Das bedeutet, dass die Summe der Grenzraten der Substitution der Konsumenten zwischen dem öffentlichen Gut und einem beliebigen Privatgut gleich der Grenzrate der Transformation zwischen diesen beiden Gütern für die Produzenten sein muss. Mit anderen Worten bedeutet dies, dass das Verhältnis des Nutzens aus dem öffentlichen Gut (der F/OS-Software) zu einem Privatgut für die Konsumenten gleich des Verhältnisses des Produktionsaufwandes zwischen diesen Gütern sein muss.

Die positive Externalität F/OSS hat aber Besonderheiten, die im Vergleich mit anderen Externalitäten, für die das Modell ursprünglich entworfen wurde, berücksichtigt werden müssen. Holtermanns Beispiele für Externalitäten entstehen im Produktionsprozess für private Güter quasi als Nebenprodukt, egal ob es sich um positive oder negative Externalitäten handelt. Sowohl der Rauch einer Fabrik (das klassische Beispiel einer negativen Externalität) als auch die Bestäubung von Pflanzen durch Bienen eines Imkers (ein beliebtes Beispiel einer positiven Externalität) entstehen als nicht direkt beabsichtigte Nebenprodukte der Produktion eines anderen Gutes.

[57] Güter, die externe Effekte verursachen.

F/OSS entsteht jedoch um ihrer selbst Willen und ist kein zufälliges Nebenprodukt[58]. Die Entwicklung geschieht mit dem klaren Ziel, am Schluss über die benötigte Software zu verfügen. Dennoch gibt es auch in der Entwicklung von F/OSS ein gewisses Maß an Zufälligkeit. Eine Software wird entwickelt, weil sie jemand benötigt, dass ein anderes Unternehmen die Software ebenfalls verwendet und sie somit ein Produktionsfaktor für das andere Unternehmen ist, ist bis zu einem gewissen Grad Zufall. Wenn dieses Unternehmen aber in die Entwicklung der Software investiert – ob finanziell oder durch direkte Beteiligung am Entwicklungsprozess –, handelt es sich nicht mehr um eine Externalität, der positive Effekt auf die Produktion wird internalisiert.
Daraus kann weiter gefolgert werden, dass es sich solange um eine positive Externalität handelt, solange das benutzende Individuum oder Unternehmen als Trittbrettfahrer handelt. Sobald es sich an der Entwicklung beteiligt, internalisiert es den positiven Effekt und tritt nicht mehr als Trittbrettfahrer auf.

5.4.5 Vergleich F/OSS – proprietäre Software

Es wurde nun gezeigt, dass und auf welche Weise F/OSS erfolgreich entwickelt werden kann. Wie die Praxis zeigt, ist F/OSS in einigen Bereichen proprietärer Software zwar überlegen, dennoch kann proprietäre Software, die gewinnbringend verkauft wird, auch neben frei zugänglicher F/OSS bestehen. Es gibt also Argumente für eine Koexistenz von F/OSS und proprietärer Software. Da Bessen dahingehend den plausibelsten Ansatz verfolgt, ist sein Modell wieder die Basis.

Bei einem relativ komplizierten F/OSS-Projekt $\tilde{m} \geq m*$ werden nicht notwendigerweise alle 2^m Arten, die Features der proprietären Software zu verwenden, unterstützt. Mit anderen Worten, wenn eine F/OSS komplexer ist als proprietäre Software gleichen Typs, heißt das nicht, dass die F/OSS alle Optionen der proprietären Software ebenso anbietet. Die Zielgruppe einer proprietären Softwarefirma wären in diesem Fall also jene Benutzer, deren Bedürfnisse mit der von der Firma entwickelten Software befriedigt werden, die an der eigenen Entwicklung der Software gescheitert sind und die sich den Preis (w) leisten können.

[58] Am ehesten könnte man F/OSS noch als zufälliges Nebenprodukt betrachten, wenn sie von Produzenten eines Komplementärgutes entwickelt wird, um die Qualität des Komplementärgutes zu erhöhen (wie in Punkt 5.4.3 besprochen wird).

Diese Benutzer haben Folgendes in den internen Entwicklungsversuch der Software investiert: $\left[p(E)V_i + (1-p(E))(V_i - w) - Ec_{FOSS}\right]$

Daraus lässt sich der optimale Preis für die Softwarefirma ableiten: $\left[wH(w)\right]$, wobei $H(w) = (1-F(w))\left(1-p\left(\tilde{E}(w)\right)\right)$. Letztere Gleichung beschreibt die Wahrscheinlichkeit, dass die Benutzer die Software kaufen und ihre interne Entwicklung erfolglos war. Der maximale Gewinn der Softwarefirma ist folglich: $\left[p(e)n\tilde{w}H(\tilde{w}) - e\cdot\left(c_0 + c2^m\right)\right]$.

Da die erste Ableitung dieses Ausdrucks nach e positiv ist, macht eine kommerzielle Softwarefirma positiven Gewinn. Ein F/OSS-Projekt und proprietäre Software können somit erfolgreich nebeneinander existieren. F/OSS-Projekte sprechen diesem Modell folgend zwei Gruppen von Benutzern an: Jene mit niedrigen Anforderungen $m \leq m*$ und niedrigem Nutzen aus der Software, die $(\tilde{w})$ nicht bezahlen können oder wollen. Und jene mit hohen Anforderungen $m > m*$. Die erste Gruppe wird wegen dem geringen Nutzen, den ihr die Software bringt, relativ wenig in die Entwicklung der Software investieren. Das ist der Grund, warum viele F/OSS-Projekte relativ komplex sind. Denn Benutzer mit hohem Nutzen aus der Software, die bereit sind viel in die Entwicklung zu investieren, haben hohe Anforderungen und werden eine Software entwerfen, die ihren Bedürfnissen entspricht.

Gibt es sowohl F/OSS als auch proprietäre Software, sinkt der Preis der proprietären Software aufgrund der Existenz der F/OSS. Denn die interne Entwicklungsinvestition, aus der sich der Preis für proprietäre Software ergibt, ist bei Koexistenz beider Varianten höher, als wenn es nur proprietäre Software gibt. Freilich wird das den Gewinn einer Softwarefirma schmälern, allerdings in verkraftbarem Ausmaß, die Softwarefirma hat hinreichenden Anreiz, den Markt nicht zu verlassen.

5.4.6 F/OSS und die öffentliche Hand

Auch wenn F/OSS in der Regel durch Private bereitgestellt wird, kommt es vor, dass die öffentliche Hand die Entwicklung von F/OSS zumindest unterstützt. Beispielsweise unterstützte die US-Regierung ein universitäres Forschungsprojekt zur Verbesserung von Linux. Auch die EU engagiert sich immer wieder in F/OSS-Projekten. Diese Unterstützung ist aber zumeist finanzieller Natur.

Wie weiter oben schon erwähnt, geht die herkömmliche Literatur über öffentliche Güter davon aus, dass es durch das Engagement des Staates zu einem Crowding-

Out des privaten Einsatzes kommen kann. Von der für die Arbeit herangezogenen Literatur geht lediglich Schmidtke umfassend darauf ein. Er[59] zeigt in seinem Paper, dass Crowding-Out nicht notwendigerweise der Fall sein muss. Dazu leitet er die (für die privaten Unternehmen gewinnmaximierenden) Bedingungen erster Ordnung $\left(\frac{\partial \pi_i}{\partial x_i}; \frac{\partial \pi_i}{\partial e_i}\right)$ erneut jeweils nach den beiden Variablen x_i und e_i ab, zieht also das totale Differential. Dadurch können die direkten und indirekten Effekte unterschieden werden. Der direkte Effekt des Investments in die Softwareentwicklung beeinflusst die Produktion des privaten Gutes direkt über die Produktion des Komplementärguts und das Investment in das öffentliche Gut der restlichen Firmen (X_{-i}, E_{-i}). Der indirekte Einfluss ändert den optimalen Produktionslevel des privaten Gutes für Firma (i), da sich der in das öffentliche Gut investierte Betrag und damit die Qualität des öffentlichen Gutes ändert.

Schmidtke modelliert das Engagement einer Regierung, die ebenfalls einen Betrag $\Delta e_R > 0$ in die Entwicklung des öffentlichen Gutes investiert. Wenn die Regierung ihr Engagement verstärkt, führt das unter bestimmten Voraussetzungen auch zu einer Verstärkung des privaten Investments, zu einem so genannten Crowding-In. Der Grenzerlös des öffentlichen Gutes muss konstant sein, seine Grenz-Produktionskosten müssen zunehmen und die Ableitung des Preises nach der Produktion des Privatgutes und dem Investment des öffentlichen Gutes darf nicht negativ sein.
Wenn die erste Bedingung erfüllt ist, hat das keinen direkten Effekt auf den Anreiz der Unternehmen ins öffentliche Gut zu investieren. Durch das Engagement der Regierung steigt aber die Qualität des öffentlichen Gutes, die Firmen haben einen Anreiz, mehr des komplementären Privatgutes zu produzieren, da durch die höhere Qualität des öffentlichen Gutes der Preis des komplementären Privatgutes steigt. Die gesteigerte Produktion des privaten Gutes hat wiederum einen indirekt positiven Effekt auf den Anreiz, ins öffentliche Gut zu investieren.
Sind diese Voraussetzungen nicht erfüllt, kann auch der gegenteilige Fall eines Crowding-Outs eintreten. Wenn der Grenzerlös des öffentlichen Gutes abnimmt, seine Grenz-Produktionskosten konstant sind und die Änderung des Regierungs-Investments nicht größer ist als das aggregierte Investment der Firmen

[59] Schmidtke (2006), S. 12ff

$\left(\sum_{i=1}^{n} e_i > \Delta e_R\right)$, kommt es zu einem Crowding-Out. Die letzte Bedingung ist auf den ersten Blick recht stark, die Änderung des Investments der öffentlichen Hand in das öffentliche Gut F/OSS müsste recht groß sein, um das gesamte vorhandene private Investment zu übersteigen.

Bei genauerer Analyse erscheint das Resultat aber durchaus plausibel. Klaus Schmidt und Monika Schnitzer argumentieren[60], dass sich der Staat, wenn er die Entwicklung von F/OSS unterstützt, wie in anderen Forschungsbereichen bei seiner Unterstützung auf die Grundlagenforschung konzentrieren soll. Angewandte Forschung, bei der die Problemstellung und das Ziel relativ klar sind, würde in ausreichendem Maß von den Marktteilnehmern betrieben werden. Das deckt sich insofern mit obigem Resultat, weil anzunehmen ist, dass das Engagement von privaten Unternehmen in der Grundlagenforschung eher gering ist. Es ist also nicht unwahrscheinlich, dass das Engagement des Staates größer ist, als das aggregierte Investment der Firmen. Ist das erfüllt, kommt es zu keinem Crowding-Out und die durch den Staat geleistete Unterstützung würde nicht durch Private geleistet.

Bei angewandter Forschung ist die Wahrscheinlichkeit, dass sich Firmen engagieren wesentlich höher. Daraus folgt, dass die Wahrscheinlichkeit, dass das Engagement des Staates kleiner ist als das aggregierte Engagement der Privatwirtschaft, steigt. Also ist bei angewandter Forschung Crowding-Out wahrscheinlicher als bei Grundlagenforschung.

5.5 Analytische Anmerkungen

Dieses Modell zeigt, dass durch die Entwicklung von F/OSS eine Pareto-Verbesserung der Situation im Softwaremarkt eintritt. Dies ist gegenüber anderen Formen der Softwareentwicklung vor allem deshalb der Fall, weil komplexe Software, die zwar nur von wenigen Benutzer(firme)n benötigt wird und deshalb auf andere Weise nicht entwickelt würde, effizient entwickelt wird. Die Software (und in der Folge Erweiterungen der Software) steht, wenn sie einmal entwickelt wurde, auch allen anderen Benutzern zur Verfügung. Dabei ist davon auszugehen, dass desto eher auf F/OSS zurückgegriffen wird, je komplexer die Anforderungen des jeweiligen Benutzers sind:

[60] Schmidt/Schnitzer (2002), S. 23ff

> „Free/Open Source will be most used by firms who have their own development capability and who have complex, specialized needs; pre-packaged software will be used by firms with simpler needs and by firms who lack development capabilities."[61]

Die Gründe für den Erfolg dieses Entwicklungsmodells liegen vor allem im massiven Sinken der Kommunikationskosten im Laufe der vergangenen 20 Jahre durch die Verbreitung des Internets. Dadurch wurde einerseits die direkte Kommunikation in der Entwicklung eines Projektes erleichtert, vor allem aber die Kommunikation über ein Projekt, da die Kosten für die Verbreitung verschiedener Entwicklungsstufen praktisch auf Null gesunken sind. Dieses Faktum trägt auch entscheidend dazu bei, dass die Trittbrettfahrerproblematik im Zusammenhang mit F/OSS nicht die gleiche Rolle spielt, wie bei der privaten Bereitstellung anderer öffentlicher Güter.
Außerdem wurde das Modell, das eine Kombination aus vorliegenden Modellen zum Thema darstellt, um zwei Punkte erweitert, die in der bisherigen Literatur unterrepräsentiert sind. Die Rolle der Bug-Reporter, Benutzer die Fehler in der Software berichten, wurde behandelt, die Frage der Trittbrettfahrer ist dadurch neu zu bewerten. Zudem wurde F/OSS als positive Externalität analysiert, für Trittbrettfahrer kann F/OSS als positive Externalität betrachtet werden, sobald sich ein Akteur in der Entwicklung engagiert, internalisiert er den positiven Effekt.
Abschließend wurde noch auf die Rolle des Staates eingegangen und gezeigt, dass sich der Staat, wenn er sich in der Entwicklung von F/OSS engagiert, auf Grundlagenforschung konzentrieren soll.

Vergleicht man die Ergebnisse dieser Arbeiten und älterer Arbeiten zum Thema private Bereitstellung öffentlicher Güter, so gibt es in den zentralen Resultaten durchaus Ähnlichkeiten. Schon Bliss und Nalebuff kommen zu dem Ergebnis, dass das Trittbrettfahrerproblem je geringer ist, desto größer die Population des Modells. Für F/OSS wird ebenfalls argumentiert, dass aufgrund der sehr großen Zahl an potenziellen Entwicklern, das Trittbrettfahrerproblem eine untergeordnete Rolle spielt. Beim Engagement des Staates in der Bereitstellung eines öffentlichen Gutes zeigt Schmidtke, dass es unter bestimmten Voraussetzungen zu einem „Crowding-In" von privatem Engagement kommen kann. Das ist gegenüber dem Modell von Bergstrom,

[61] Bessen (2005), S. 23

Blume und Varian, die lediglich von einem „Crowding-Out“ (das Schmidtke freilich auch behandelt) ausgehen, eine Weiterentwicklung.

6 Die Motivation der Entwickler

Wie bereits besprochen ist der große Pool an potenziellen Entwicklern einer der Gründe für den Erfolg von F/OSS. Durch die Offenheit des Quellcodes sind F/OSS-Projekte für praktisch alle interessierten Entwickler zugänglich. Eine der meist gestellten Fragen im Zusammenhang mit F/OSS ist die Frage nach der Motivation der Entwickler, die an Software-Projekten mitarbeiten, ohne eine direkte Gegenleistung oder finanzielle Abgeltung erwarten zu können. Während das Modell dieser Arbeit überwiegend Fälle behandelte, in denen sich Unternehmen an der Entwicklung von F/OSS beteiligten, betrachtet dieser Abschnitt Privatpersonen, die zur Projektentwicklung beitragen.

Die Entwicklergemeinde eines F/OSS-Projekts setzt sich meist aus verschiedenen Gruppen zusammen. Die Zusammensetzung ist von Projekt zu Projekt verschieden, bei vielen Projekten sind auch nicht alle der beschriebenen Entwicklergruppen engagiert, andere Projekte werden von einer Gruppe dominiert.
Ein Teil der Entwickler sind Angestellte eines Unternehmens, das sich in der Entwicklung des jeweiligen Projektes engagiert. Sie arbeiten im Grunde nicht anders als Entwickler von proprietärer Software und erhalten Lohn von einem Unternehmen, das sich von der zu entwickelnden Software einen wie auch immer gearteten Nutzen erhofft. Dann gibt es Entwickler, die ebenfalls Lohn erhalten, der aber in direktem Zusammenhang mit diesem Projekt steht. Sie werden für die Arbeit an dem spezifischen Projekt von einem Unternehmen oder einer Organisation für ihre Rolle in der Projektentwicklung bezahlt, sind aber abseits dieser Rolle dem Unternehmen nicht verpflichtet. Diese Art des Engagements gibt es in unterschiedlichem Umfang.
Die wahrscheinlich größte Gruppe an Entwicklern ist jene, die ohne Bezahlung und direkte Abgeltung ihres Zeiteinsatzes an der Entwicklung von F/OSS mitarbeiten. Deren Motivation steht im Mittelpunkt dieses Abschnitts, ist sie doch abseits von herkömmlich monetären Arbeitsanreizen zu suchen. Letztere sind aber auch bei F/OSS-Entwicklern nicht gänzlich auszuschließen. Wie zu sehen sein wird, spielen indirekte monetäre Anreize zum Teil eine Rolle.

6.1 Die Partizipation von privaten Entwicklern

Das umfassende Engagement von Softwareentwicklern in der Entwicklung von F/OSS abseits ihrer beruflichen Verpflichtungen ist einer der wichtigsten Gründe für die signifikante Verbreitung von F/OSS. Die typischen ökonomischen Erklärungsmodelle greifen für das freiwillige, vorderhand unentgeltliche Engagement zu kurz. Warum arbeitet eine Gruppe von bedeutender Größe ohne Bezahlung an unzähligen Projekten? Dieser Frage haben sich schon einige empirische Arbeiten zum Thema F/OSS angenommen, die signifikantesten Resultate sollen hier diskutiert werden. Wie zu sehen sein wird, hat diese Form der Informationsgüterproduktion auch Auswirkungen auf andere Bereiche als die Softwareindustrie. Außerdem sind ähnliche Phänomene auch bei der Bereitstellung anderer Informationsgüter zu beobachten.

In der Literatur werden die Anreize für die Entwickler in verschiedene Gruppen unterteilt. Das erscheint sinnvoll, da es in der heterogenen Gruppe der F/OSS-Entwickler auch heterogene Gründe für das Engagement geben wird. Eine mögliche Unterscheidung ist zwischen Anreizen für den persönlichen Nutzen und gesellschaftlichen Anreizen[62]. Zur ersten Gruppe würden finanzielle Anreize (die auch indirekt sein können) und Selbstverwirklichung zählen, zur zweiten Anreize wie eine Leistung zu Gunsten der Gesellschaft oder die politische Überzeugung.
Eine weitere Unterscheidung, die häufig zu finden ist, ist zwischen intrinsischen und extrinsischen Motiven[63]. Diese Unterscheidung kommt aus der Psychologie. Extrinsische Motive stehen für Anreize durch äußeren Zwang oder äußere Belohnung, intrinsische Motive beruhen „auf der Befriedigung aus der Betätigung heraus“[64]. Dabei kann es zu Überschneidungen zwischen den Unterscheidungsarten kommen, extrinsische Motive können beispielsweise auch Anreize für den persönlichen Nutzen sein.
Die finanziellen Anreize können auch als Anreize im Zusammenhang mit der eigenen beruflichen Situation angesehen werden. Direkte finanzielle Einkünfte aus dem Engagement in F/OSS-Projekten sind zwar die Ausnahme, indirekte Einkünfte sind aber wahrscheinlich. Da alle Änderungen und Fortschritte eines Projektes mit dem

[62] Sie steht im Mittelpunkt der Arbeit von Haruvy, Wu und Chakravarty (2005). Die Autoren verschmelzen die beiden Bereiche aber zu einer singulären Erklärung der Motivation.
[63] Auch Stein (2006, S. 106f.) verwendet diese Unterscheidung.
[64] Stein (2006), S. 107

dafür verantwortlichen Autor dokumentiert werden, kann ein Entwickler seine Leistungen für die F/OSS-Community praktisch in den Lebenslauf übernehmen. Sie können ein Signal an potenzielle Arbeitgeber über die Fähigkeiten des Entwicklers sein. Das Engagement kann also mit individuellen Bildungsausgaben verglichen werden. Je mehr sich ein Entwickler in F/OSS-Projekten engagiert und umso erfolgreicher er beim Entwickeln ist, desto größer wird sein Bekanntheitsgrad in der Community und desto höher ist sein später zu erwartendes Einkommen.

Außerdem vergrößert die Arbeit in F/OSS-Projekten in gewisser Weise die Berufserfahrung eines Entwicklers. Bei Studenten oder Schülern (die einen signifikanten Anteil an F/OSS-Entwicklern stellen) hat die Beschäftigung quasi die Wirkung eines die Ausbildung begleitenden Praktikums, ist also mit Bildungsausgaben vergleichbar. Aber auch für andere Entwickler ist der Erfahrungsgewinn relevant, da die Arbeit an F/OSS die generelle Problemlösungskompetenz auf Softwareebene erhöht.

Von der Warte einer Entwicklungsfirma aus sind die Dokumentationen der F/OSS-Projekte ein Indikator für die Fähigkeiten potenzieller Beschäftigter. Die Entwickler, die sich dabei als am fähigsten erweisen, sind für die Unternehmen am interessantesten, für sie werden auch die höchsten Preise, sprich Löhne, bezahlt. Dabei dient das Engagement nicht nur als Signal für eventuelle Anstellungen bei Entwicklungsfirmen, auch einzelne Entwicklungsaufträge von Firmen können derart lukriert werden.

Der entscheidende Anreiz zum Start eines F/OSS-Projektes ist oftmals ein Do-It-Yourself-Gedanke. So war der ursprüngliche Zweck des GNU-Projekts, auf dem das F/OSS-Betriebssystem Linux basiert, ein frei zugängliches Betriebssystem zu schaffen, weil ein solches schlicht noch nicht existierte. Unzählige F/OSS-Projekte sind jedoch entstanden – und entstehen immer noch –, weil ein Entwickler eine Anwendung benötigt, diese zu entwickeln beginnt, sie der interessierten Öffentlichkeit via Internet zur Verfügung stellt und sich Mitarbeit und Verbesserungen von anderen Benutzern oder Entwicklern erhofft, die die gleiche oder eine ähnliche Anwendung benötigen. Vielen Entwicklern macht die Arbeit an neu zu entwickelnden oder zu verbessernden Programmen schlicht und einfach auch Spaß. Somit zählt auch persönlicher Lustgewinn zu den potenziellen Anreizen zur Teilnahme an F/OSS-Projekten.

Damit in Zusammenhang steht oftmals ein anderer Community-interner Aspekt. Der erfolgreiche Start oder die Mitarbeit an einem Projekt bringen Ansehen innerhalb der

Entwicklergemeinschaft. Die Mitarbeit an Projekten zeitigt für die Entwickler auch nicht-monetäre Einkünfte wie Selbstverwirklichung, Genugtuung oder Selbstvertrauen.

Auf der anderen Seite stehen Anreize, die den Entwicklern nicht direkte persönliche Einkünfte bringen, aber ebenfalls mit Selbstverwirklichung oder Genugtuung umschrieben werden können. Viele Entwickler engagieren sich bei F/OSS-Projekten auch aufgrund ihrer politischen Überzeugung. Auch dabei spielen mannigfaltige Gründe eine Rolle: Die freie Verbreitung von Wissen, der Kampf gegen Softwarefirmen oder das Recht auf frei zugängliche Software. Dabei ist interessant, dass die Entwickler durch die Bereitstellung eines öffentlichen Gutes zum Teil Aufgaben des Staates übernehmen, folgt man der herkömmlichen ökonomischen Theorie. Sie sehen ihre freiwillige Arbeit als Dienst an der Gesellschaft und schlüpfen so quasi freiwillig in die Rolle des Staates, der diese Aufgabe der Theorie folgend eigentlich übernehmen müsste. Wichtig ist, dass die verschiedenen aufgezeigten Anreize jeweils nur für einen Teil der Entwickler gelten, teilweise nebeneinander existieren und in der Community zum Teil kontrovers diskutiert werden.

6.2 Zum Typus des F/OSS-Entwicklers

Es wäre übertrieben zu behaupten, dass die Gruppe der F/OSS-Entwickler ein wissenschaftlich gut erforschtes Phänomen ist. Eine Handvoll Arbeiten, die Rückschlüsse über die Entwickler und ihre Beweggründe zulassen, gibt es aber. Dass die Entwickler mehrheitlich relativ jung sind, erscheint, da es sich bei F/OSS in der heutigen Form um ein relativ junges Phänomen handelt, logisch, wird aber auch von den empirischen Daten unterstützt. So verortet Stein[65] in seinem Überblick über verschiedene Studien[66] den Großteil der Teilnehmer an F/OSS-Projekten in der Altersgruppe zwischen 20 und 40 Jahren. Wobei ein leichter Schwerpunkt in der ersten Dekade zu vermuten ist. Dort liegt auch das durchschnittliche Einstiegsalter

[65] Stein (2006), S.100f.

[66] Er konzentriert sich im Großen und Ganzen auf die Studien von Lakhani/Wolf (2003), Hars/Ou (2001), sowie die FLOSS-Studien von je einer europäischen, nordamerikanischen und japanischen Universität (für Details siehe Literaturliste). Lakhani/Wolfs Sample umfasst 707 Befragte, Hars/Ous 81 Befragte und die FLOSS-Studie kam in Europa auf 2784 valide Teilnehmer, in den USA auf 1588 und in Japan auf 547. Es erscheint mir sinnvoll hier überblicksartig die Ergebnisse verschiedener Studien zu verwenden. So erhält man zwar keine exakten Ergebnisse aber aussagekräftige Bandbreiten, in welcher Größenordnung sich die jeweilige Größe bewegt.

eines Entwicklers in die Projektentwicklung, bei circa 22 Jahren.
Bei den Entwicklern von F/OSS handelt es sich mit großer Mehrheit um Männer. Ein Frauenanteil von 5 Prozent ist bereits als hoch einzustufen. Interessant ist, dass der durchschnittliche Anteil von Programmiererinnen normalerweise signifikant höher (rund 20 bis 30 Prozent) liegt. Dabei lebt der größere Teil in irgendeiner Form von Partnerschaft, die FLOSS-Studien kommen auf rund 40 Prozent Singles, zwischen 20 und 30 Prozent Verheiratete und 30 bis 40 Prozent in Partnerschaft lebende. Interessant ist die Tatsache, dass der Großteil der befragten Entwickler irgendeiner Form von bezahlter Beschäftigung nachgeht und die Arbeit an F/OSS nur nebenbei erledigt. Der Anteil der Studenten und Schüler liegt in der Regel unter 25 Prozent, ist aber im Vergleich zur Repräsentation in der gesamten Gesellschaft doch überproportional. Der Großteil der Beschäftigten geht einem Beruf nach, der im Zusammenhang mit Software steht. So sind die drei häufigsten Berufe neben Student in der europäischen FLOSS-Studie Softwareentwickler, Programmierer und IT-Berater. Dabei ist ein relativ großer Anteil der Befragten dieser Studie (14 Prozent) selbständig.
Mit diesen Aussagen konsistent scheint die Beobachtung, dass ein überdurchschnittlich hoher Anteil der Entwickler einer hohen Bildungsstufe angehört, zwischen circa 43 und 70 Prozent liegt der Anteil der Akademiker. Dabei muss es sich freilich nicht zwingend um eine, wie man annehmen könnte, computer-spezifische Ausbildung handeln. Ein guter Teil der Entwickler hat sich seine Fähigkeiten selbst beigebracht, der Anteil der Autodidakten oder jener ohne Programmierausbildung liegt je nach Studie zwischen 35 und 40 Prozent. Das Einkommen der meisten Entwickler ist nicht überdurchschnittlich hoch. Über die Hälfte der befragten Entwickler verdient nicht mehr als 3000 Dollar beziehungsweise Euro.

Da es sich beim Großteil der Entwickler um anderweitig Beschäftigte oder Studenten handelt, entspricht für die meisten Teilnehmer an F/OSS-Projekten das Zeitbudget, dass für die Arbeit an den Projekten aufgewendet wird, dem eines Hobbies oder Nebenberufes. Der Anteil jener, die bis zu zehn Stunden wöchentlich für F/OSS aufwenden liegt zwischen 70 und 80 Prozent. Andererseits ist die Gruppe jener, die mehr als 20 Stunden aufwenden – was nebenberufliche Beschäftigung doch deutlich übersteigt –, mit 10 bis 15 Prozent ebenfalls erstaunlich hoch. Ein guter Teil der F/OSS-Entwickler ist neben der Entwicklung von F/OSS noch mit der Entwicklung

von proprietärer Software beschäftigt. Das wird zum Großteil auf beruflicher Basis – ob direkt angestellt oder über Projektaufträge – geschehen. Dabei ist der Zeitaufwand in der Regel ungleich größer als für F/OSS. Von den Entwicklern, die auch proprietäre Software entwickeln, tun das 55 bis 60 Prozent mehr als 20 Stunden wöchentlich.

Der typische Entwickler konzentriert sich in der Regel auf wenige Projekte, an denen er mitarbeitet. So haben 72 Prozent der europäischen FLOSS-Studie, seit sie sich in der F/OSS-Community engagieren, nicht zu mehr als fünf Projekten beigetragen. Zum Zeitpunkt der Studie waren mehr als 56 Prozent der Befragten in ein oder zwei Projekten engagiert. Folgt man der FLOSS-Studie, ist die Community sehr breit aufgestellt, zwei Drittel der Beteiligten hatte zum Zeitpunkt der Studie bereits Erfahrung als Leiter, Administrator oder Koordinator eines F/OSS-Projekts.

Weniger erstaunlich ist die Tatsache, dass die meisten Entwickler ihren Lebensmittelpunkt in industrialisierten Ländern haben, ist doch dort die Verbreitung der Hardware, für die eine Software entwickelt wird (Computer, Handy, Server, etc.), am höchsten. So ist in jeder betrachteten Studie eine große Mehrheit von über 75 Prozent in Westeuropa oder Nordamerika zuhause.

6.3 Die Gründe für das Engagement

Wie weiter oben beschrieben, gibt es verschiedenste Gründe, die Entwickler zur Teilnahme an F/OSS-Projekten inspirieren. Die Befragung von Hars/Ou aus dem Jahr 2001 kommt zum Ergebnis, dass die wichtigsten Motive für die Entwickler „Selbstbestimmung“ mit knapp 80 Prozent Zustimmung und „Humankapitalbildung“ mit knapp 90 Prozent Zustimmung sind. Wobei letzteres wohl auch mit Bildungsausgaben umschrieben werden könnte, die Entwickler sehen die aus dem Engagement gezogene Erfahrung und Weiterentwicklung der eigenen Fähigkeiten als sehr wichtig an. Recht wenig Zustimmung (16,5 Prozent) bekommt das Motiv „Altruismus“, also der hehre Dienst an der Gesellschaft. Ebenfalls relativ unbedeutend ist der potenzielle „Verkauf von Produkten“ mit nur knapp 14 Prozent Zustimmung.

Etwas anders sieht das Bild in der Studie von Lakhani/Wolf aus, hier haben Motive, die als altruistisch bezeichnet werden können, wie „Quellcode sollte frei sein“ und „Verpflichtung der F/OSS-Bewegung etwas zurückzugeben“ Zustimmungsraten von

rund einem Drittel. Bei den extrinsischen Motiven hat das Motiv „Programmierfähigkeiten entwickeln“ (rund 40 Prozent) die höchste Zustimmung, ein Motiv, das auch ins Muster „Humankapitalbildung“ passt.

In der europäischen FLOSS-Studie sind ähnliche Motive dominant, die höchste Zustimmung für den Einstieg in die Entwicklergemeinschaft haben das „Lernen und Entwickeln neuer Fähigkeiten“ (knapp 80 Prozent) und diese „Fähigkeiten und Wissen auszutauschen“ (knapp 50 Prozent). Das sind auch die wichtigsten Beweggründe, der F/OSS-Community treu zu bleiben. Geringe Bedeutung haben Motive wie die „Verbreitung nicht marktfähiger Software“, „Reputation in der Community“ und „um Geld zu verdienen“ (jeweils rund 10 Prozent).

Der Vergleich mit der japanischen Studie liefert einige interessante Details. Während die beliebtesten Gründe für den Einstieg und Verbleib ähnliche Zustimmung erreichen, gibt es bei anderen Motiven erhebliche Unterschiede. Intrinsische und zum Teil altruistische Motive wie „um an einer neuen Form von Kooperation teilzuhaben“ oder „Software soll frei sein“ erreichen in Europa ansehnliche Zustimmung von über 35 Prozent, gehören in Japan aber zu den unbedeutendsten Motiven. Ähnlich verhält es sich mit einem weiteren altruistischen oder vielmehr politischen Motiv, „die Macht großer Softwarekonzerne zu brechen“.

Nach den Ergebnissen der US-amerikanischen FLOSS-Studie sind derartige Motive überdurchschnittlich wichtig. „Der F/OSS-Community etwas zurückzugeben“ und „Software soll frei modifizierbar sein“ sind für je rund 78 Prozent „wichtig“ oder „sehr wichtig“. Weniger wichtig sind Motive wie der „Reiz an der Fehlerbehebung und Problembeseitigung“ (für 40 Prozent „wichtig“ oder „sehr wichtig“), die „Notwendigkeit Fehler in einer Software zu beheben“ (53 Prozent) oder „Interesse an der Arbeitsweise eines Programms“ (55 Prozent).

6.4 Resümee

Wie zu Beginn angenommen wurde, sind für den einzelnen Entwickler verschiedenste Gründe für die Beteiligung an F/OSS-Projekten ausschlaggebend. Bei einer derart großen heterogenen Gruppe sind typische Motive für das Engagement praktisch nicht auszumachen. Die Fülle von verschiedenen, teils widersprüchlichen Motiven widerlegt gängige Vorurteile über F/OSS-Entwickler. Auch wenn beispielsweise die Mehrheit der Entwickler auch in ihrem beruflichen Leben mit

Software zu tun hat, trifft das bei weitem noch nicht auf alle Entwickler zu. So sind zuverlässige, zusammenfassende Aussagen kaum möglich. Es erscheint als erwiesen, dass für viele Entwickler Gründe wie „Weiterbildung“ oder „Humankapitalbildung“ eine gewichtige Rolle spielen. Andererseits gibt es auch eine signifikante Gruppe von Teilnehmern, die der Kampf gegen Softwarekonzerne motiviert. Somit kann dieser Abschnitt lediglich einen Einblick in die heterogene Welt der F/OSS-Entwickler bieten, er ist nicht zu mehr geeignet, als eine Idee davon zu geben, was die Teilnehmer an F/OSS-Projekten antreibt, und so zum Erfolg der F/OSS-Projekte beiträgt.

7 Zusammenfassung und abschließende Bemerkungen

Diese Arbeit hatte zum Ziel, die private Bereitstellung des öffentlichen Gutes Free/Open Source Software zu analysieren. In der ökonomischen Theorie wird davon ausgegangen, dass öffentliche Güter in der Regel nicht vom Markt bereitgestellt werden. Der Staat muss, so die Theorie, für die Bereitstellung dieser Güter Sorgen. Eine der wenigen Ausnahmen dieser Regel ist das öffentliche Gut Free/Open Source Software. Die Gründe liegen in speziellen Eigenschaften dieses Gutes, die in der Arbeit zu erklären versucht wurde.

Zuerst wurde der Begriff der Free/Open Source Software definiert und abgegrenzt und die Entwicklung dieser Form der Softwareentwicklung nachgezeichnet. Das ist insofern wichtig, da auch andere Softwareformen viele der behandelten Eigenschaften haben, aber nur F/OSS alle diese Eigenschaften in sich vereint und deshalb besondere Betrachtung verdient. Im Anschluss wurde das Gut Software definiert. Dieses Gut ist schon per se ein Besonderes und lässt sich kaum mit herkömmlichen privaten oder öffentlichen Gütern vergleichen. Bei Software handelt es sich um ein Informationsgut, genauer um ein digitales Gut.
Das besondere an digitalen Gütern ist, dass sie praktisch kostenfrei und ohne Qualitätsverlust reproduziert werden können. Sämtliche Produktionskosten sind für die erste Einheit zu leisten. Ab der zweiten Einheit sinken die Produktionskosten auf Null. Zudem sind digitale Güter nicht-rivalisierend, unendlich erweiterbar, diskret, unräumlich und rekombinant. Dabei verdient sich vor allem die Eigenschaft der Diskretheit genauere Betrachtung, denn Software ist sowohl diskret als auch stetig. Einerseits ist Software unteilbar, andererseits ist Software nie fertig, oder „ganz“, die Entwicklung von Software ist häufig ein laufender Prozess, der nie abgeschlossen wird.

Im nächsten Kapitel wurde der Begriff des öffentlichen Gutes erklärt und argumentiert, warum es sich bei F/OSS um ein öffentliches Gut handelt. Es müssen einige Voraussetzungen erfüllt sein, um F/OSS konsumieren zu können, da es sich dabei aber um Voraussetzungen handeln, die nicht direkt mit F/OSS in Verbindung stehen, sind sie einer Definition von F/OSS als öffentlichem Gut nicht im Weg.
In der Folge wurden allgemeine Arbeiten zum Thema private Bereitstellung

öffentlicher Güter betrachtet und auf ihre Verwertbarkeit für die Fragestellung der Arbeit untersucht. Die Ergebnisse, die in diesen Arbeiten erzielt wurden, sind zum Teil auch für die private Bereitstellung von F/OSS gültig. Allerdings gelten einzelne Annahmen nicht mehr und die Forschung zu F/OSS kommt zu teils widersprechenden Resultaten.

Das vierte Kapitel ist das zentrale Kapitel dieser Arbeit. Aus bestehenden Arbeiten zum Thema wurde ein Modell zusammengefügt, das verschiedene Formen der Softwareentwicklung analysiert und zeigt, warum es durch die Entwicklung von F/OSS zu einer Pareto-Verbesserung kommt. Dabei wurde auf die Trittbrettfahrerproblematik genau eingegangen und die Rolle der Bug-Reporter, ein Punkt der in der bisherigen Literatur nicht beachtet wird, beleuchtet. Außerdem wurde die Bedeutung des Modulcharakters der Software für die Entwicklung und die Entwicklung von Software als Komplementärgut besprochen.
Ein weiterer Punkt, auf den die vorliegende Literatur zum Thema wenig eingegangen ist, F/OSS als positive Externalität wurde in das Modell eingefügt. Dieser Punkt ist im Zusammenhang mit der Frage nach der Trittbrettfahrerproblematik zu sehen, F/OSS ist für Trittbrettfahrer als positive Externalität zu betrachten, sobald sich ein Trittbrettfahrer in der Entwicklung von F/OSS engagiert, internalisiert er den externen Effekt. Außerdem wurde im Modell die Konkurrenzfähigkeit von F/OSS im Vergleich mit proprietärer Software analysiert und besprochen, inwieweit ein Engagement der öffentlichen Hand bei der Entwicklung von F/OSS Sinn macht.
Im letzten Kapitel wurden die Motive von Entwicklern, sich bei der Entwicklung von F/OSS zu engagieren, besprochen. Dabei wurde auf die verschiedenen Formen der Motive und die Typen der Entwickler, die an F/OSS-Projekten teilnehmen, eingegangen. Zudem wurden Resultate verschiedener empirischer Studien zu diesem Thema diskutiert. Da es sich bei Softwareentwicklern um eine heterogene Gruppe handelt, sind pauschale Aussagen über die Motive schwer möglich. „Weiterbildung“ dürfte für viele Entwickler ein wichtiges Motiv sein, für einen kleinen, aber doch signifikanten Teil spielt der Kampf für eine bessere Welt oder gegen die Macht von Softwarekonzernen eine wichtige Rolle.

Diese Arbeit zeigt, dass die private Bereitstellung des öffentlichen Gutes Free- und Open Source Software durch die besonderen Eigenschaften des Gutes ermöglicht wird. Dabei gibt es vielfältige Gründe, warum F/OSS privat bereitgestellt wird. F/OSS

steht zumeist allerdings nicht in direkter Konkurrenz zu proprietärer Software. Die beiden Softwarearten befriedigen unterschiedliche Bedürfnisse unterschiedlicher Benutzer und existieren deshalb erfolgreich nebeneinander. Dieser Befund ist allerdings stark von der Beschaffenheit der jeweiligen Programme abhängig. Es gibt auch F/OSS-Programme, die viel Wert auf Benutzerfreundlichkeit legen und als direkte Konkurrenten zu proprietärer Software auftreten. Beispielsweise sieht der Mitgründer von Mozilla Europe, jene Organisation, die den erfolgreichsten F/OSS-Webbrowser Firefox vertreibt, den Grund für den Erfolg von Firefox im besonderen Augenmerk auf Benutzerfreundlichkeit.[67]

Die Art und Weise, wie F/OSS entwickelt wird, ist eine besondere Produktionsweise, die bisherige Bereitstellungsmechanismen zwar nicht revolutionieren wird, aber vor allem bei Informationsgütern[68] in Zukunft an Bedeutung gewinnen könnte.

[67] http://www.itpro.co.uk/124832/love-and-usability-drive-firefox-success, 11.2.2009

[68] Erwähnt sei das Beispiel Wikipedia. Diese Online-Enzyklopädie funktioniert nach einem ähnlichen Prinzip.

8 Literaturverzeichnis

- **Aghion**, P. und J. **Tirole**: “The Management of Innovation”, in: Quarterly Journal of Economics, 109, 1994, S. 1185-1209
- **Bergstrom**, T., L. **Blume** und H. **Varian**: “On the Private Provision of Public Goods” in: Journal of Public Economics, 29, 1986, S. 25-49
- **Bessen**, James: “Open Source Software: Free Provision of Complex Public Goods”, Research on Innovation Working Papers, Boston University School of Law, 2005
- **Bliss**, C. und B. **Nalebuff**: “Dragon-slaying and Ballroom Dancing: The Private Supply of a Public Good” in: Journal of Public Economics, 25, 1984, S. 1-12
- **Brousseau**, E. and N. **Curien**: „Internet and Digital Economics“, Cambridge University Press, Cambridge, 2007
- **Ghosh**, R. A., R. **Glott**, B. **Krieger** und G. **Robles**: „Free Software Developers: Who, how and why“ in: Soete, L.: “The economics of the digital society”, Elgar, Cheltenham, 2005, S. 152-184
- **Hann**, I.-H., J. **Roberts**, S. **Slaughter** und R. **Fielding**: “An Empirical Analysis of Economic Returns to Open Source Participation”, Working Paper, Carnegie-Mellon University, 2004
- **Hars**, A. und S. **Ou**: “Working for Free? - Motivations of Participating in Open Source Projects,” International Journal of Electronic Commerce, 6(3), 2002, S. 25–39.
- **Haruvy**, E., F. **Wu** und S. **Chakravarty**: “Incentives for Developers’ Contributions and Product Performance Metrics in Open Source Development: An Empirical Exploration”, IIMA Working Paper, 2005
- **Holtermann**, S. E.: “Externalities and Public Goods” in: Economica, vol. 39, 153, 1972, S. 78-87
- **Johnson**, J. P.: „Open Source Software: Private provision of a public good“ in: Journal of Economic & Management Strategies, 4/2002, S. 637-662
- **Lakhani**, K. und R. **Wolf** : “Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects,” MIT Sloan Working Paper No. 4425-03, 2003
- **Lerner**, J. und J. **Tirole**: “The economics of technology sharing: open source and beyond”, Nber Working Papers Series, No.10956, Cambridge (MA), 2004

- **Lerner**, J. und J. **Tirole**: “The Simple Economics of Open Source”, NBER Working Paper Series, No. 7600, 2000
- **Palfrey**, T. R. und H. **Rosenthal**: “Participation and the Provision of Discrete Public Goods: A Strategic Analysis”, Journal of Public Economics, 24, 1984, S. 171-193
- **Quah**, D.: “Digital Goods and the New Economy”, CEP Discussion Paper Nr. 563, London School of Economics, 2002
- **Schiff**, A.: “The Economics of Open Source Software: A Survey of the Early Literature”, Review of Network Economics, 2002, vol. 1, 1, S. 66-74
- **Schmidt**, K. M. und M. **Schnitzer**: „Public Subsidies for Open Source? Some Economic Policy Issues of the Software Market”, Working Paper, University of Munich, 2002
- **Schmidtke**, R.: "Private provision of a complementary public good", Working Paper, University of Munich, 2006
- **Sebald,** G.: „Offene Wissensökonomie“, Verlag für Sozialwissenschaften, Wiesbaden, 2008
- **Shapiro**, C. und H. **Varian**: „Information Rules“, Harvard Business School Press, Boston (MA), 1999
- **Stein**, A.: „Die Open-Source-Bewegung“, VDM Müller, Saarbrücken, 2006
- **Varian**, H.: “Grundzüge der Mikroökonomik”, Oldenbourg, München, 2001

8.1 Online-Literaturverzeichnis

- **Auburn University:** http://www.auburn.edu/~johnspm/gloss/public_goods, 5.3.2008
- **FLOSS-Studien:**
  - EU: http://infonomics.nl/FLOSS/report/FLOSS_Final4.pdf, 12.11.2008; http://infonomics.nl/FLOSS/report/FLOSS_Final5all.pdf, 12.11.2008
  - Japan: http://oss.mri.co.jp/floss-jp/report_en.html, 14.11.2008
  - USA: http://stanford.edu/group/floss-us/report/FLOSS-US-Report.pdf, 14.11.2008
- **GNU Operating System:** http://www.gnu.org/philosophy/free-sw.html, 04.11.2008
- **IT Pro:** http://www.itpro.co.uk/124832/love-and-usability-drive-firefox-success,

11.2.2009

- **Stern (Nachrichtenmagazin):** http://www.stern.de/computer-technik/telefon/:Open-Handset-Alliance-Google-Android-Handy/601768.html, 20.2.2009
- **Wikipedia**: http://en.wikipedia.org/wiki/Open_Source_history, 23.04.2008; http://en.wikipedia.org/wiki/Open_Handset_Alliance, 20.2.2009

Anhang

Abstract (deutsch)

Diese Arbeit betrachtet das Phänomen der Free und Open Source Software als privat bereitgestelltes öffentliches Gut. Nach einer Begriffsklärung und einer Abgrenzung von Software im Bereich der Informationsgüter, wird der Forschungsgegenstand in Bezug zum theoretischen Konzept der öffentlichen Güter gebracht. Die Entwicklung und Bereitstellung von Free und Open Source Software wird im Hauptteil der Arbeit anhand eines formalen Modells beschrieben. Dieses Modell entstand aus verschiedenen vorliegenden Modellen, die sich mit Teilaspekten der Thematik auseinandersetzen, und enthält neue Aspekte der Thematik. Zudem wurde in der Arbeit anhand verschiedener empirischer Studien auf die Motive der Entwickler, die zur privaten Bereitstellung dieses öffentlichen Gutes beitragen, eingegangen.

Abstract (english)

This thesis regards Free and Open Source Software as a privately provided public good. Starting with a disambiguation and a classification of Software in the field of information goods, the thesis sets the object of research into context with the theoretical concept of public goods. The main part of the thesis describes the development and provision of Free and Open Source Software by means of a formal model. This model was generated from various existing models which deal with different aspects of the subject and contains new viewpoints of the topic. Furthermore the thesis contains a section dealing with the motives of the developers contributing to the private provision of this public good.

Lebenslauf

Persönliche Daten:

Matthias Nagl
Geboren am 16.01.1983 in Oberndorf bei Salzburg
Staatsbürgerschaft: Österreich

Schulische Ausbildung:

- 09/1989 – 07/1993 Volksschule Salzburg-Liefering I
- 09/1993 – 06/2002 Privatgymnasium der Herz-Jesu-Missionare in Salzburg-Liefering

Universitärer Werdegang:

- 10/2003 – 01/2009 Studium der Volkswirtschaftslehre an der Universität Wien
- Seit 10/2003 Studium der Politikwissenschaft an der Universität Wien

Sonstige Ausbildung:

- 07/2006 – 10/2006 Lehrredaktion der „Wiener Zeitung“